FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

A Comprehensive Analysis of the cf2 Argumentation Semantics:

From Characterization to Implementation

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der technischen Wissenschaften

by

Sarah Alice Gaggl
Registration Number 0026566

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Privatdoz.Dr. Stefan Woltran
Ao.Univ.Prof.Dr. Uwe Egly

The dissertation has been reviewed by:

(Privatdoz.Dr. Stefan Woltran) (Prof.Dr. Pietro Baroni)

Wien, 13.02.2013

(Sarah Alice Gaggl)

Erklärung zur Verfassung der Arbeit

Sarah Alice Gaggl
Severingasse 8/6, 1090 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasserin)

Acknowledgments

I dedicate this thesis to my brother Lukas who taught me to pursue a goal even though it is very hard and always to look on the bright side of life.

The support of many persons in my life enabled me to finally complete this thesis. Therefore I would like to dedicate some lines to say thank you. First of all I am very grateful to my parents, Albin and Elisabeth Gaggl, and my brother Lukas who always supported me during my studies and who enabled me to go my way.

Special thanks go to Ülkü who always believed in me, sometimes I think more than I believed in myself. Without her support, diversion and patience I would have never got so far. Furthermore, Cornelia, Lili and Sybille where ready to help and listen whenever I needed it. I know this was not easy all the time.

I am very glad that Stefan Woltran was my supervisor during the last years. I was able to learn a lot from him. He guided my work as well as he gave me space and time to make my own experiences and decisions. When it came to review this thesis I could always count on receiving his feedback within a few days which facilitated my work extremely. I could not have had a better advisor.

I was lucky to meet and get in contact with many colleagues which resulted in fruitful, inspiring and also enjoyable conversations. I could never mention everybody so I give a (possibly incomplete) unordered list of the persons which had most influence on me during the last years: Magdalena Widl, Uwe Egly, Dov Gabbay, Wolfgang Dvořák, Katrin Seyr, Francesca Toni, Johannes Wallner, Paul E. Dunne, Reinhard Pichler, Pietro Baroni, Gerd Brewka, Peter Schüller, Sebastian Rudolph, Hans Tompits, Sanjay Modgil, Thomas Eiter, Torsten Schaub, Thomas Krennwallner, Georg Gottlob, Federico Cerutti, Nysret Musliu.

Finally I want to thank the Vienna Science and Technology Fund (WWTF) for the funding of my work through project ICT08-028.

Abstract

Argumentation is one of the major fields in Artificial Intelligence (AI). Numerous applications in diverse domains like legal reasoning, multi-agent systems, social networks, e-government, decision support and many more make this topic very interdisciplinary and lead to a wide range of different formalizations. Out of them the concept of abstract Argumentation Frameworks (AFs) is one of the most popular approaches to capture certain aspects of argumentation. This very simple yet expressive model has been introduced by Phan Minh Dung in 1995. Arguments and a binary *attack* relation between them, denoting conflicts, are the only components one needs for the representation of a wide range of problems and the reasoning therein.

Nowadays numerous semantics exist to solve the inherent conflicts between the arguments by selecting sets of "acceptable" arguments. Depending on the application, acceptability is defined in different ways. Some semantics are based on the idea to defend arguments against attacks, while others treat arguments like different choices and the solutions stand for consistent sets of arguments. A systematic analysis of these semantics on a theoretical and practical level is indispensable for the development of competitive systems. This includes a complete complexity analysis to develop appropriate algorithms and systems, the verification of the behavior on concrete instances as well as the identification of possible redundancies for specific semantics to simplify the frameworks.

In this thesis we exemplify such an analysis on the $cf2$ semantics which does not require to defend arguments against attacks but is based on a decomposition of the framework along its strongly connected components (SCCs). This allows to treat cycles in a more sensitive way than others and to overcome some problems which arise with odd- and even-length cycles. Due to the quite complicated definition of this semantics it has not been studied very intensively.

To facilitate further investigation steps we first introduce an alternative characterization of the $cf2$ semantics. Then we propose a small modification of this semantics to overcome a particular problematic behavior on specific instances which results in the sibling semantics $stage2$. After a complete complexity analysis and the investigation of equivalences for these two semantics, we apply the obtained results on two different implementation methods, namely the reduction-based approach of answer-set programming and the direct implementation in terms of labeling-based algorithms.

Kurzfassung

Argumentation ist ein wichtiges Forschungsfeld der Künstlichen Intelligenz. Zahlreiche Anwendungen in den Bereichen Legal Reasoning, Multi-Agenten Systeme, Soziale Netzwerke, E-Government, Decision Support und viele weitere machen dieses Gebiet sehr interdisziplinär und führen zu einer Vielzahl von Formalisierungen. Dabei hat sich das Konzept der *Abstract Argumentation Frameworks* (AFs) zu einem der beliebtesten Ansätze entwickelt. Dieses relativ einfache aber sehr ausdrucksstarke Model wurde im Jahre 1995 von Phan Minh Dung eingeführt. Dabei stellen Argumente und eine binäre Relation zwischen den Argumenten, genannt *Attacken*, die einzigen Komponenten dar um eine große Anzahl von Problemstellungen zu behandeln.

Mittlerweile existieren sehr viele Semantiken um die Konflikte zwischen den Argumenten zu lösen und *zulässige* Mengen von Argumenten auszuwählen. Abhängig von der jeweiligen Anwendung genügen diese Semantiken unterschiedlichen Anforderungen. Einige basieren auf dem Konzept, dass Argumente gegen Attacken verteidigt werden, wohingegen bei anderen die Lösungen durch konsistente Mengen von Argumenten gegeben sind. Eine systematische Analyse dieser Semantiken, sowohl auf theoretischer als auch auf praktischer Ebene, ist unabdingbar um wettbewerbsfähige Systeme zu entwickeln. Dazu gehören die Komplexitätsanalyse um geeignete Algorithmen zu designen, die Untersuchung des Verhaltens an konkreten Instanzen und auch die Identifizierung von möglichen Redundanzen zur Vereinfachung der Frameworks.

In dieser Arbeit werden wir eine solche Analyse anhand der $cf2$ Semantik durchführen. Diese Semantik basiert auf einer Zerlegung des Frameworks entlang seiner stark zusammenhängenden Komponenten, wobei das Konzept der Verteidigung der Argumente gegen Attacken vernachlässigt wird. Die $cf2$ Semantik hat den speziellen Vorteil, dass sie mit Zyklen ungerader Länge sensibler umgehen kann als andere Semantiken. Dadurch kann die $cf2$ Semantik auch für AFs eingesetzt werden, die sowohl Zyklen gerader als auch ungerader Länge aufweisen. Da jedoch die Definition dieser Semantik relativ kompliziert ist wurde sie bis jetzt noch nicht besonders ausführlich in der Literatur behandelt.

Um die weitere Untersuchung zu erleichtern führen wir eine alternative Charakterisierung der $cf2$ Semantik ein. Dann stellen wir eine geringfügige Abänderung vor, um ein gewisses problematisches Verhalten an speziellen Instanzen zu beheben, welche zu der verwandten $stage2$ Semantik führt. Nach einer umfassenden Komplexitätsanalyse und der Untersuchung von Äquivalenzen für diese beiden Semantiken, wenden wir die erlangten Resultate für zwei unterschiedlichen Implementierungsmethoden an, nämlich in Form von Answer-Set Programming und von Algorithmen die auf der Berechnung von *Labelings* basieren.

Contents

CHAPTER 1

Introduction

1.1 Argumentation in Artificial Intelligence

The concept of Argumentation has been studied within the last years very intensively. In 1995, Phan Minh Dung first introduced the formalism of abstract Argumentation Frameworks (AFs), a very simple yet expressive approach to capture certain aspects of argumentation (see [37]). Arguments and a binary relation between them, denoting conflicts, are the only components one needs for the representation and reasoning of a wide range of problems. Dung already provided in [37] many semantics to solve the inherent conflicts between the arguments. Furthermore, he investigated the relation of abstract AFs to Default Logic, Defeasible Logic and Logic Programming (LP). Although the research on dialectic and argumentation can be traced back to the classical Greek philosopher Plato, Dung inspired with his work many researchers to further studies. One can say that he gave the theoretical starting point for a whole research field (see [21] for an overview).

The research done in abstract argumentation ranges from the representation and modeling of different scenarios [5, 76, 101], the creation of new semantics [9, 12, 17, 25, 96] and extensions of the framework [13, 19, 20, 32, 77, 92, 93], to a more general view of the problematic by distancing from the abstract level and taking the whole argumentation process into account [29]. This process includes three major steps:

1. Representation/generation of the arguments;

2. Identification of the conflicts between the arguments;

3. Solving the conflicts via selecting acceptable subsets of arguments.

In abstract argumentation one only takes the arguments and the relation between them into account by abstracting from the internal structure of the arguments. Hence, the focus is only on step 3.

Argumentation Semantics

The solution of the inherent conflicts is performed on a semantical level, where one has many different options to select acceptable sets of arguments depending on the specific requirements. The basic principle of all argumentation semantics is to obtain conflict-free sets of arguments. Traditional argumentation semantics build on the concept of admissible sets, i.e. sets where each argument attacking an argument in the set is also attacked by the set. Most of the prominent semantics count to this category, like preferred, stable, complete and grounded, just to mention some of them.

However, recent investigations [8, 12, 20, 23] showed that in in certain situations admissible-based semantics do not provide satisfying results. For instance the appearance of odd-length cycles and in particular self-attacking arguments as a special case of them, have a strong and sometimes undesired influence on the computation of solutions. None of the admissible-based semantics is able to select arguments of such a cycle as accepted, and moreover, they sometimes reject arguments just because they are attacked by an argument contained in an odd-length cycle.

$cf2$ Semantics. One way of overcoming these effects is to detach from the need of defending the arguments but to see the arguments as different choices, where a solution of the conflicts can be a maximal consistent set of arguments. The so called naive-based semantics do not rely on the notion of defense, thus one can accept both, arguments in an odd-length cycle, as well as arguments attacked by an odd-length cycle. Besides the naive (maximal conflict-free) semantics also stage [96] and $cf2$ semantics count to this category[1].

The $cf2$ semantics has been introduced in [6] and later in [12], Baroni et al. introduced a general SCC-recursive schema for argumentation semantics, based on a decomposition of the framework along its strongly connected components (SCCs), which also contained the $cf2$ semantics. The $cf2$ semantics has some significant advantages by treating cycles in a more sensitive way than others. Hence, it overcomes some problems which arise with odd- and even-length cycles.

1.2 Main Contributions

Due to the quite complicated definition of the $cf2$ semantics it is not as well studied as others. Therefore, the main focus of the thesis will be on the investigation of this semantics. In the following we sketch the state-of-the-art of relevant problems arising in the course of this investigation and describe the main contributions of this thesis. We start with an alternative characterization of $cf2$ semantics.

Alternative Characterization. The initial motivation for modifying the definition of $cf2$ arose from the difficulties to encode the semantics in answer-set programming (ASP). It turns out to be rather cumbersome to represent $cf2$ semantics directly within ASP. This is due to the fact that the original definition involves a recursive computation of different sub-frameworks. Therefore, we shift the need of recursion from generating sub-frameworks to the concept of recursively

[1] One special candidate is stable semantics which is both admissible- and naive-based.

component defeated arguments, which can be captured via a fixed-point operator $\Delta_{F,S}$ for an AF F and a set S. Then, we construct an *instance* of F with respect to $\Delta_{F,S}$ and check whether the set S is a naive extension of both, F and the instance of F. In other words, this allows us to characterize $cf2$ semantics using only linear recursion.

With the alternative characterization at hand we are able to design the corresponding ASP encodings, by first guessing a naive extension S and then checking whether S is a naive extension of the respective instance of the given AF F. Furthermore, the novel characterization of $cf2$ facilitates further investigation steps.

$stage2$ Semantics. Although there have been pointed out several advantages of $cf2$ in the literature as mentioned above, also this semantics shows undesired behavior in some situations. In particular the evaluation of odd-cycle-free AFs e.g. if even-length cycles occur, is now questionable [64, 69]. On the other side, stage semantics [96] can also handle odd-length cycles and does not change the behavior of odd-cycle-free AFs. The disadvantages of stage semantics are that very basic properties are not satisfied, for example the skeptical acceptance of unattacked arguments, i.e. the weak reinstatement property [8] is violated. While naive-based semantics seem to be the right candidates when the above described behavior of admissible-based semantics is unwanted, there are several shortcomings with existing approaches, as mentioned above. To overcome those problems we propose a new semantics combining concepts from $cf2$ and stage semantics, which we name $stage2$. Thus, we use the SCC-recursive schema of $cf2$ semantics and instantiate the base case with stage semantics. It turns out, that the novel $stage2$ semantics overcomes the shortcomings of both $cf2$ and stage semantics. As $cf2$ and $stage2$ semantics are closely related, we include the novel $stage2$ semantics in the continuative investigation and compare the obtained results between $cf2$ and $stage2$ semantics.

Computational Complexity. An important issue in the analysis of argumentation semantics has always been the study of computational complexity [36, 39, 48, 50]. Whereas for most of the argumentation semantics and the respective reasoning problems, an extensive complexity analysis exists, the $cf2$ semantics has been neglected in this context. Such an analysis is indispensable for the implementation of efficient algorithms and systems. Therefore, we will study the standard reasoning problems of the argumentation semantics $cf2$ and $stage2$, namely (i) verification, (ii) credulous acceptance, (iii) skeptical acceptance, and (iv) existence of a non-empty extension. Moreover, we provide an analysis of possible tractable fragments [34, 54, 55] which can help to improve the performance for easy instances of in general hard problems. In particular we consider acyclic AFs, even-cycle free AFs, bipartite AFs and symmetric AFs.

Notions of Equivalence. As argumentation is a dynamic reasoning process it is of specific interest to know the effects additional information may cause with respect to a specific semantics. Oikarinen and Woltran [84] identified kernels that eliminate redundant attacks of AFs and introduced the concept of *strong equivalence*: two AFs are strongly equivalent w.r.t. a semantics σ (i.e. they provide the same σ-extensions no matter how the two AFs are simultaneously extended), if their σ-kernels coincide. Different notions of equivalence have been studied in [18, 84] for most of the semantics.

To complete the picture we analyze standard and strong equivalence for $cf2$ and $stage2$ semantics. Interestingly it turns out that for both of them, strong equivalence coincides with syntactic equivalence. Thus, there are no redundant attacks at all, which means that every part of the AF has a potential influence on the evaluation of the extensions. We make this particular behavior more explicit by defining a new property for argumentation semantics, the *succinctness property*. If a semantics σ satisfies the succinctness property, then for every framework F, all its attacks contribute to the evaluation of at least one framework F' containing F.

Implementation. In order to evaluate argumentation frameworks and to compare the different semantics, it is desirable to have efficient systems at hand which are capable of dealing with a large number of argumentation semantics. As argumentation problems are in general intractable, which is also the case for $cf2$ and $stage2$ semantics, developing dedicated algorithms for the different reasoning problems is non-trivial. A promising way to implement such systems is to use a reduction method, where the given problem is translated into another language, for which sophisticated systems already exist. It turned out that the declarative programming paradigm of *Answer-Set Programming* (ASP) is especially well suited for this purpose (see [95] for an overview). The attempt to use logic programming to encode argumentation problems is not new, Dung already highlighted this approach in [37] as well as Nieves et al. in [81, 82, 85], Wakaki and Nitta in [99] and Egly et al. in [57, 59].

In this work we follow the ASPARTIX approach as introduced by Egly et al., where the semantics are encoded within a fixed query and the concrete AF to process is provided as the input for the program. This has several advantages, as the input AF can be changed easily and dynamically without translating the whole formula, which simplifies the answering of questions like "What happens if I add this new argument?" Furthermore, the modularity of ASP programs allows to easily extend, change and reuse parts of the encodings. On the performance side one can observe that advanced ASP solvers like clasp, claspD, DLV, Cmodels, Smodels, IDP, or SUP are nowadays able to deal with large problem instances, see [24].

As mentioned above, the alternative characterization allows to encode the $cf2$ (resp. $stage2$) semantics with the widely used Guess&Check methodology for ASP programs. Moreover, recent developments like the `metasp` front-end [70] for the ASP-system `gringo/claspD` allow to optimize and simplify complicated encodings, like the ones needed for reasoning problems located at the second level of the polynomial hierarchy, as it is the case for $stage2$ semantics.

Besides the reduction based approach, one can of course also design algorithms which directly compute the desired solution of the reasoning problems. In this content we consider here a labeling-based approach [30, 97]. In contrast to the traditional extension-based approach, so called labelings distinguish two kinds of unaccepted arguments, those which are rejected by the extension and those which are neither rejected nor accepted. This distinction is interesting from a logic perspective but has also proven to be useful for algorithmic issues. Although there has already been defined a labeling for $cf2$ semantics in [14], we present here a slightly different one which reflects the behavior of these semantics more explicitly. Besides the definition of labelings for $cf2$ (resp. $stage2$) semantics we provide labeling based algorithms to compute all solutions of an AF in terms of labelings.

Finally, we point out that although the ASPARTIX system does not require that the user is an ASP expert, still one needs to have an ASP solver available. Therefore, we developed a web front-end of ASPARTIX which is freely accessible from any standard web browser[2]. This tool makes use of the ASP encodings but the concrete procedure is completely hidden from the user. Besides the computation of all extensions for a wide range of semantics (including $cf2$ and $stage2$), the tool offers a graphical representation of the input framework and the solutions.

To summarize, this work is dedicated to provide more insights into argumentation semantics, exemplified on the $cf2$ semantics to make argumentation systems more competitive for the future.

1.3 Structure of the Thesis

This thesis is organized as follows.

- In Chapter 2, *Background of Abstract Argumentation*, we introduce all Dung semantics as well as stage, semi-stable, ideal, eager, resolution-based grounded and of course the $cf2$ semantics. Then in Section 2.2 we point out some special properties of the semantics and we classify them w.r.t. their subset-relation. Regarding $cf2$ semantics we will illustrate the problematic behavior on frameworks with cycles of length ≥ 6. In Section 2.3 we recall the evaluation criteria introduced in [8] which are of interest for the naive-based semantics, and give the respective results for the introduced semantics.

- Chapter 3 is dedicated to the *Alternative Characterization* of $cf2$ semantics. After introducing some preliminaries, we first give the $cf2$ definition based on the computation of a set of recursively component defeated arguments $\mathcal{RD}_F(S)$. Then we prove that the set $\mathcal{RD}_F(S)$ can be captured via a fixed-point operator $\Delta_{F,S}$. This allows us to characterize $cf2$ semantics using linear recursion only. We conclude the chapter with an analysis where we point out some advantages of the introduced alternative characterization.

- In Chapter 4, *Incorporating Stage Semantics in the SCC-recursive Schema*, we introduce the novel $stage2$ semantics, which uses the SCC-recursive schema of $cf2$ and instantiates the base case with stage semantics. Furthermore, we also formulate $stage2$ semantics with the characterization introduced in Chapter 3. In Section 4.2 we compare $stage2$ with the other naive-bases semantics, namely with $cf2$, stage and stable semantics, and give the respective relations in terms of subset-inclusion. Then, in Section 4.3 we investigate $stage2$ semantics regarding the evaluation criteria introduced before. Finally, in Section 4.4 we summarize the obtained results of this chapter.

- In Chapter 5 we concentrate on the analysis of *Computational Complexity* of $cf2$ and $stage2$ semantics. After a short recapitulation of the basic concepts of computational complexity we investigate the complexity of the main reasoning problems for argumentation semantics. In Section 5.3 we consider tractable fragments for $cf2$ and $stage2$ semantics, and conclude in Section 5.4.

[2] `http://rull.dbai.tuwien.ac.at:8080/ASPARTIX`

- In Chapter 6 we study different *Notions of Equivalence*. In Section 6.1 we start with introducing the necessary background on standard and strong equivalence followed by the definition of the succinctness property. Then, in Section 6.2 we consider $cf2$ and $stage2$ semantics, as well as their base semantics naive and stage, in terms of standard equivalence. In Section 6.3 we characterize strong equivalence for $cf2$ and $stage2$ semantics, as well as for naive and stage. Finally, in Section 6.4 we compare the semantics with respect to strong equivalence and we shortly discuss strong equivalence for symmetric frameworks.

- In Chapter 7 we turn to the *Implementation* of $cf2$ and $stage2$ semantics. After introducing the basic concepts of ASP, we give the ASP encodings for $cf2$ followed by the ones for $stage2$ semantics. For the latter one we start with the saturation encodings for stage semantics, as it is the base semantics of $stage2$. Thanks to the modularity of ASP we can then put the different parts together and obtain the desired encodings. Besides the more involved saturation method we also mirror a novel optimization technique which makes use of the `metasp` front-end for the ASP-system `gringo/claspD`. This allows us to formulate ASP encodings for $stage2$ (resp. stage) which are shorter and easier to understand than the saturation encodings, without the loss of performance. In Section 7.2 we give two algorithms for $cf2$ and $stage2$ semantics which are based on the computation of labelings. Finally, in Section 7.3 we briefly present the web-application of ASPARTIX, before we conclude the implementation part in Section 7.4.

- Finally, in Chapter 8 we summarize the contributions of this thesis and make a critical reflection of the obtained results. In Section 8.3 we discuss related work and in Section 8.4 we point out some possible future directions.

1.4 Publications

The growing interest on argumentation led to many publications on different platforms. Articles from the field of argumentation are under the top citations at *Artificial Intelligence* journal, the *International Conference on Computational Models of Arguments* (COMMA) is held every second year since 2006, the first *International Workshop on the Theory and Applications of Formal Argumentation* (TAFA) was co-located at the *International Joint Conference on Artificial Intelligence* (IJCAI) in 2011, recently two textbooks appeared, namely *Elements of Argumentation* in 2008 [22] and *Argumentation in Artificial Intelligence* in 2009 [90].

Parts of this thesis have been published at international conferences, workshops, journal papers and in a book chapter. In the following we shortly sketch the contributions.

The alternative characterization of $cf2$ presented in Chapter 3 has been introduced first at the *COMMA'10* conference [67] where the article was awarded with the *Best Student Paper Award*. The investigation of notions of equivalence of $cf2$, stage and naive semantics has been published at the *ECSQUARU'11* conference [68].

The article in the *Journal of Logic and Computation* [69] gives a more detailed description of the alternative characterization of $cf2$, the analysis of notions of equivalence w.r.t. $cf2$, stage and naive semantics, the first definition of the succinctness property, as well as the complexity analysis of *cf2* semantics as described in Section 5.2. Furthermore, the questionable behavior of *cf2* on longer cycles has been pointed out with a hint to instantiate the base case with stage semantics instead of naive semantics.

The *stage2* semantics as described in Chapter 4 has been formally introduced and presented at *NMR'12* [44], where the authors were awarded with the *Best Student Paper Prize*. This article also contains the complexity analysis of the standard reasoning problems for *stage2* semantics as presented in Section 5.2. Then, in the article presented at *COMMA'12* [45] the analysis of computational aspects of *cf2* and *stage2* semantics has been continued. In particular the investigation of tractable fragments as described in Section 5.3 and the labeling based algorithm for *cf2* as in Section 7.2 is included there.

The general ASPARTIX approach has been first presented at the *ICLP'08* [57] and at the *ASPOCP'08* workshop [58]. An extensive version of the ASP encodings for argumentation frameworks has then been presented in the journal *Argument and Computation* [59]. Some of the techniques we used for the encodings in Section 7.1, like the saturation, the ordering and stratified programs has been described there in detail. The ASP encodings for *cf2* semantics from Section 7.1 have been presented in [67]. At *INAP'11* [52] we presented how to use the `metasp` optimization front-end for argumentation semantics located at the second level of the polynomial hierarchy like preferred, semi-stable and stage semantics. We used this technique also to simplify the encodings for resolution-based grounded semantics, and the article also contains the standard saturation encodings of stage semantics. A performance evaluation of the traditional saturation encodings versus the simplified ones is also included.

Regarding *stage2* semantics we only sketched how the encodings can be built in the article presented at *COMMA'12*. The detailed encodings for *stage2*, both the saturation and the `metasp` ones, are newly described in this thesis. The web application of ASPARTIX has been presented at the software demonstration session at *COMMA'10*, and the general ASPARTIX approach has been presented at the *ICLP Doctoral Consortium 2010* [66] and at the poster session of the *ACAI Summer School 2009*.

In the Chapter *The Added Value of Argumentation* of the book *Agreement Technologies* the need for a benchmark library for abstract argumentation has been pointed out together with several ideas how this can be achieved [53]. We will shortly discuss this matter in Section 8.2.

Finally, we mention that an outline of this thesis has been presented at the *KR Doctoral Consortium 2012*.

CHAPTER 2

Background of Abstract Argumentation

In this chapter we first introduce the basics of abstract argumentation, the semantics we need for further investigations and some properties of the semantics we are mainly interested in this work, the $cf2$ semantics.

Abstract argumentation frameworks have been first introduced by Dung [37] in 1995. It is a very simple but also very powerful formalism to reason over conflicting knowledge. The syntax only consists of a set of statements called *arguments* and a binary relation between them, the *attacks* denoting the conflicts between the arguments. As we are on the abstract level, we do not concentrate on the internal structure of the arguments but only on their relation to each other. This means we assume the framework has been instantiated correctly by an expert. The following definitions of abstract argumentation frameworks and the semantics are based on [12, 37, 96].

Definition 1. *An* argumentation framework (AF) *is a pair $F = (A, R)$, where A is a finite set of arguments and $R \subseteq A \times A$. The pair $(a, b) \in R$ means that a attacks b. A set $S \subseteq A$ of arguments* attacks b *(in F), if there is an $a \in S$, such that $(a, b) \in R$. An argument $a \in A$ is* defended *by $S \subseteq A$ (in F) iff, for each $b \in A$, it holds that, if $(b, a) \in R$, then S attacks b (in F).*

In this work we require that the AFs are finite, as it is the case in most of the theoretical investigations on abstract argumentation. However, in practice this is not always guaranteed. Recent approaches dealing with infinite AFs are the *argumentation frameworks with recursive attacks* (AFRAs) [15, 16] and the *extended argumentation frameworks* (EAFs) [78].

In the following we fix some notations we will use throughout the thesis. AFs $F_1 = (A_1, R_1)$ and $F_2 = (A_2, R_2)$ are called *disjoint* if $A_1 \cap A_2 = \emptyset$. Moreover, the union between (not necessarily disjoint) AFs is defined as $F_1 \cup F_2 = (A_1 \cup A_2, R_1 \cup R_2)$. For an AF $F = (A, R)$, we will use the notations $A(F)$ and A_F to address the arguments of F. When we speak about attacks we will use $R(F)$ as well as R_F.

Such AFs are typically represented as a directed graphs as in the following example.

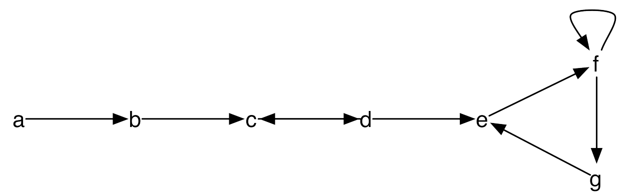

Figure 2.1: The argumentation framework F from Example 1.

Example 1. *Consider the AF $F = (A, R)$, consisting of the set or arguments $A = \{a, b, c, d, e, f, g\}$ and the attack relation $R = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, f), (f, f), (f, g), (g, e)\}$ as illustrated in Figure 2.1.* $\diamond$

2.1 Semantics of Abstract Argumentation

The inherent conflicts between the arguments are solved by selecting subsets of arguments, where a semantics σ assigns a collection of sets of arguments to an AF F. The basic requirement for all semantics is that none of the selected arguments attack each other; these sets are then called *conflict-free*.

Definition 2. *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is said to be* conflict-free *(in F), if there are no $a, b \in S$, such that $(a, b) \in R$. We denote the collection of sets which are conflict-free (in F) by $cf(F)$. A set $S \subseteq A$ is maximal conflict-free or* naive*, if $S \in cf(F)$ and for each $T \in cf(F)$, $S \not\subset T$. We denote the collection of all naive sets of F by $naive(F)$. For the empty AF $F_0 = (\emptyset, \emptyset)$, we set $naive(F_0) = \{\emptyset\}$.*

Clearly, all argumentation semantics are based on conflict-free sets. In the following we give the definitions of the semantics introduced by Dung in [37], which are all *admissible-based*, i.e. sets where each argument in the set is defended by the set.

Definition 3. *Let $F = (A, R)$ be an AF. A conflict-free set $S \in cf(F)$ is said to be*

- *a* stable *extension (of F), i.e. $S \in stable(F)$, if each $a \in A \setminus S$ is attacked by S (in F);*
- *an* admissible *extension (of F), i.e. $S \in adm(F)$, if each $a \in S$ is defended by S (in F);*
- *a* preferred *extension (of F), i.e. $S \in pref(F)$, if $S \in adm(F)$ and for each $T \in adm(F)$, $S \not\subset T$;*
- *a* complete *extension (of F), i.e. $S \in compl(F)$, if $S \in adm(F)$ and for each $a \in A$ defended by S (in F), $a \in S$ holds;*
- *a* grounded *extension (of F), i.e. the unique set $S \in grd(F)$, if S is the least (w.r.t. set inclusion) complete extension (of F).*

Among the semantics from Definition 3, the grounded extension is the only one which has a *unique status approach*. This means that for every AF F, $|grd(F)| = 1$ and it can also be defined as the least fixed-point (lfp) of the following characteristic function $\mathcal{F}_F(S)$.

Definition 4. *Given an AF $F = (A, R)$ and let $S \subseteq A$. The characteristic function $\mathcal{F}_F : 2^A \to 2^A$ of F is defined as*

$$\mathcal{F}_F(S) = \{x \in A \mid x \textit{ is defended by } S\}.$$

To illustrate the different behavior of the introduced semantics we have a look at the AF from Example 1.

Example 2. *Consider the AF $F = (A, R)$ as in Figure 2.1. Then, the above defined semantics yield the following extensions.*

- $naive(F) = \{\{a, d, g\}, \{a, c, e\}, \{a, c, g\}\}$*;*
- $stable(F) = \emptyset$*, this is the only semantics where it can happen that there does not exist any extension;*
- $adm(F) = \{\{\}, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}\}$*, note that the empty set is always an admissible extension;*
- $pref(F) = \{\{a, c\}, \{a, d\}\}$*;*
- $compl(F) = \{\{a\}, \{a, c\}, \{a, d\}\}$*;*
- $grd(F) = \{\{a\}\}$.

$\diamond$

After Dung's 1995 paper, many more semantics and also extensions of the framework have been introduced. In the following we recall the semantics which attracted most interest and are in some relevance to our further investigations. We start with the *stage semantics* introduced first by Verheij [96] in 1996 and reinvestigated by Caminada [28]. The stage semantics was the first approach where the arguments in an acceptable set do not need to defend against all attacks. Thus, it is the first semantics not based on admissible sets but as we will see later, on naive sets. In the following we call those semantics *naive-based*. To this end we define the *range* of a set of arguments as follows.

Definition 5. *Let $F = (A, R)$ and $S \subseteq A$. We define the* range *of S (w.r.t. R) as*

$$S_R^+ = S \cup \{b \mid \exists a \in S, \textit{ s. t. } (a, b) \in R\}.$$

Then, the stage extensions of an AF are the conflict-free sets with maximal range.

Definition 6. *Let $F = (A, R)$ and $S \in cf(F)$, then S is a* stage *extension (of F), i.e. $S \in stage(F)$, if there is no $T \in cf(F)$ with $T_R^+ \supset S_R^+$. We denote the collection of all stage extensions of F by $stage(F)$.*

The stage extensions of the AF from Example 1 are $stage(F) = \{\{a, d, g\}, \{a, c, e\}, \{a, c, g\}\}$. One special feature of stage semantics is that they can select arguments out of odd-length cycles and they can also accept arguments which are attacked by an odd-length cycle. A special case of an odd-length cycle is a self attacking argument. Whereas admissible-based semantics, which are all semantics defined in [37], are based on the notion of defense, they are never able to accept neither an argument out of an odd-length cycle nor an argument attacked by an odd-length cycle. We are going to demonstrate this special behavior later when we discuss the properties of the related semantics.

The next semantics we consider is the *semi-stable semantics*, introduced by Caminada [25] in 2006 and investigated also in [42]. Semi-stable semantics are located in-between stable and preferred semantics, in the sense that each stable extension of an argumentation framework F is also a semi-stable extension of F, and each semi-stable extension of F is a preferred extension of F. However, in general both inclusions do not hold in the opposite direction. In contrast to the stable semantics, semi-stability guarantees that there exists at least one extension (in case of finite AFs). We use the definition given in [42].

Definition 7. *Let $F = (A, R)$ be an AF, and a set $S \subseteq A$. A set S is a* semi-stable extension *of F, if $S \in adm(F)$ and for each $T \in adm(F)$, $S^+_R \not\subset T^+_R$. We denote the collection of all semi-stable extensions of F by $semis(F)$.*

Remember, the AF from Example 1 has no stable extension but two preferred extensions, namely $\{\{a, c\}, \{a, d\}\}$. For semi-stable semantics we obtain one extension, hence $semis(F) = \{\{a, d\}\}$ and as stated above, $semis(F) \subseteq pref(F)$ holds.

The *ideal semantics*, defined by Dung, Mancarella and Toni in 2007 [38], selects the maximal (w.r.t. $\subseteq$) admissible set which is contained in every preferred semantics, hence the ideal semantics also satisfies the unique status approach.

Definition 8. *Let $F = (A, R)$ be an AF. A set $S \subseteq A$ is an* ideal set *of F, if $S \in adm(F)$ and for each $T \in pref(F)$, $S \subseteq T$ holds. Then, S is the (unique)* ideal extension *of F, i.e. $S \in ideal(F)$ if it is the maximal (w.r.t. $\subseteq$) ideal set of F.*

The idea of ideal reasoning has been continued by Caminada in 2007 [26], where the preferred extensions have been replaced by semi-stable extensions. Then, an *eager extension* is the maximal (w.r.t. $\subseteq$) admissible set which is contained in every semi-stable extension.

Definition 9. *Let $F = (A, R)$ be an AF. A set $S \in adm(F)$ is an* eager set*, if for any $T \in semis(F)$, $S \subseteq T$ holds. Then, S is the (unique)* eager extension *i.e. $S \in eager(F)$ if it is the maximal (w.r.t. $\subseteq$) eager set.*

For the AF F from Example 1 we obtain, $ideal(F) = \{\{a\}\}$ and $eager(F) = \{\{a, d\}\}$. The ideal reasoning is less skeptical then the grounded semantics and it does not always coincide with the intersection of all preferred (resp. semi-stable) extensions as exemplified in the following example given in [38].

Example 3. *Consider the AF F of Figure 2.2. The preferred extensions of F are $pref(F) = \{\{b, d, f\}, \{b, c, f\}\}$, so $\{b, f\} = \{b, d, f\} \cap \{b, c, f\}$, but $ideal(F) = \{\{b\}\}$ since $\{b, f\}$ is not an admissible extension of F.* $\diamond$

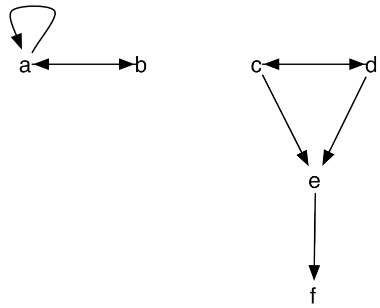

Figure 2.2: The argumentation framework F from Example 3.

In 2011, Dvořák, Dunne and Woltran generalized the notion on ideal acceptability to further semantics [51] .

The last semantics we introduce here is the *resolution-based grounded semantics* which has been defined within a the family of resolution-based semantics in [17].

Definition 10. *A resolution $\beta \subset R$ of an $F = (A, R)$ contains exactly one of the attacks (a, b), (b, a) if $\{(a, b), (b, a)\} \subseteq R$, $a \neq b$, and no further attacks. The union of all resolutions of an AF F will be denoted as $res(F)$. A set $S \subseteq A$ is a* resolution-based grounded extension *of F, i.e. $S \in grd^*(F)$ if*

(i) there exists a resolution β such that $S = grd((A, R \setminus \beta))$;[3] *and*

(ii) there is no resolution β' such that $grd((A, R \setminus \beta')) \subset S$.

This semantics has been defined, because none of the other semantics satisfies all evaluation criteria proposed in [8]. We are going to discuss some of the evaluation criteria in Section 2.3. In contrast to the grounded extensions, the resolution-based grounded semantics belongs to the *multiple status approach*, hence an AF can have more than one resolution-based grounded extension.

We consider the AF F from Example 1 which had one mutual attack between the arguments c and d. Thus, there are two resolutions of F, i.e. $res(F) = \{\beta_1, \beta_2\}$ with $\beta_1 = \{(c, d)\}$ and $\beta_2 = \{(d, c)\}$. The resolution-based grounded extensions of F are then computed as follows.

- $grd((A, R \setminus \beta_1)) = \{a, d\} = S_1$;
- $grd((A, R \setminus \beta_2)) = \{a, c\} = S_2$.

Both sets fulfill Condition (ii) of Definition 10, as $S_1 \not\subset S_2$, $S_2 \not\subset S_1$ and there are no further resolutions of F. Thus, we obtain $grd^*(F) = pref(F) = \{\{a, c\}, \{a, d\}\}$. Recall, the (single) grounded extension of F is the set $\{a\}$.

The second example we consider for resolution-based grounded semantics is the AF F of Example 3, consisting of two mutual attacks and the empty set as its grounded extension. For

[3] Slightly abusing notation, we use $grd(F)$ for denoting the unique grounded extension of F.

this AF we obtain $res(F) = \{\beta_1, \beta_2, \beta_3, \beta_4\}$ with $\beta_1 = \{(b, a), (d, c)\}$, $\beta_2 = \{(b, a), (c, d)\}$, $\beta_3 = \{(a, b), (c, d)\}$ and $\beta_4 = \{(a, b), (d, c)\}$. Then, the grounded extension of the modified frameworks are as follows.

- $grd((A, R \setminus \beta_1)) = \{c, f\} = S_1$;
- $grd((A, R \setminus \beta_2)) = \{d, f\} = S_2$;
- $grd((A, R \setminus \beta_3)) = \{b, d, f\} = S_3$;
- $grd((A, R \setminus \beta_4)) = \{b, c, f\} = S_4$.

It follows, $S_1 \subset S_4$ and $S_2 \subset S_3$. Thus we finally obtain $grd^*(F) = \{\{c, f\}, \{d, f\}\}$.

SCC-recursive Schema and *cf2* Semantics

The *cf2* semantics has been originally defined by Baroni and Giacomin in 2003 [6] as an approach to solve several problems which arise for frameworks with odd-length cycles. Later in 2005 they defined a general SCC-recursive schema for argumentation semantics [12] where the *cf2* semantics is also involved. The authors in [12] describe a general schema which captures all Dung semantics. The SCC-recursive schema is based on a recursive decomposition of an AF along its strongly connected components. In this work we only concentrate on one special case of this schema, the *cf2* semantics.

As mentioned before, all admissible-based semantics, i.e. semantics which build on the concept of admissible sets, cannot accept arguments out of an odd-length cycle. We already introduced stage semantics as the first semantics based on naive sets. On the basis of this requirement one can classify the semantics into admissible-, and naive-based semantics. All Dung semantics fall into the category of admissible-based semantics, whereas naive, stage as well as *cf2* and *stage2* (introduced next and in Chapter 4) count to the naive-based semantics. Only stable semantics falls into both groups as we show in the following lemma.

Lemma 1. *For any AF $F = (A, R)$ such that $stable(F) \neq \emptyset$, $stable(F) \subseteq adm(F)$ and $stable(F) \subseteq naive(F)$.*

Proof. We recall the definition of stable extensions: For any AF $F = (A, R)$ a conflict-free set S is a stable extension of F, if each $a \in A$ is attacked by S in F. It is easy to see that each stable extension S is also an admissible extension. S is conflict-free and all arguments not belonging to S are attacked by S, thus all arguments in S are defended by S which meets the definition of admissible sets.

To show $stable(F) \subseteq naive(F)$, we assume towards a contradiction there exists a set $S \in stable(F)$ such that $S \notin naive(F)$. Clearly S is conflict-free, so there exists a set $T \in cf(F)$ such that $S \subset T$. Then, there is an argument $a \in T$ such that $a \notin S$. From S being a stable extension we know that each argument not contained in S is attacked by S, thus there exists a $b \in S$ with $(b, a) \in R$. As $S \subset T$ it follows $b \in T$ which is a contradiction to $T \in cf(F)$. Thus, we showed each $S \in stable(F)$, is also a naive set of F. $\square$

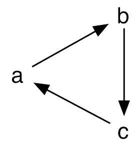

Figure 2.3: The argumentation framework F from Example 4.

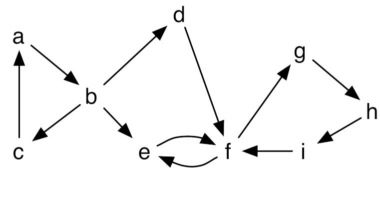

Figure 2.4: The argumentation framework F from Example 5.

Example 4. *Consider the AF $F = (A, R)$ as depicted in Figure 2.3. Then, the empty set is the only extension which would be accepted by admissible-based semantics like preferred, complete or grounded. The stable semantics does not even accept the empty set. On the other side, the naive sets are $\{a\}$, $\{b\}$ and $\{c\}$.* $\diamond$

In the following we introduce the naive-based semantics *cf2* which is based on a decomposition along the strongly connected components (SCCs) of an AF. Hence, we require some further formal machinery.

Definition 11. *A directed graph is called* strongly connected *if there is a path from each vertex in the graph to every other vertex of the graph. By $SCCs(F)$, we denote the set of* strongly connected components *of an AF $F = (A, R)$, i.e. sets of vertices of the maximal strongly connected sub-graphs of F; $SCCs(F)$ is thus a partition of A.*

Moreover, for an argument $a \in A$, we denote by $C_F(a)$ the component of F where a occurs in, i.e. the (unique) set $C \in SCCs(F)$, such that $a \in C$.

Example 5. *We consider the framework $F = (A, R)$ with $A = \{a, b, c, d, e, f, g, h, i\}$ and $R = \{(a, b), (b, c), (c, a), (b, d), (b, e), (d, f), (e, f), (f, e), (f, g), (g, h), (h, i), (i, f)\}$ as illustrated in Figure 2.4. F has three SCCs, namely $C_1 = \{a, b, c\}$, $C_2 = \{d\}$ and $C_3 = \{e, f, g, h, i\}$. The argument g belongs to C_3, thus $C_F(g) = C_3$.* $\diamond$

It turns out to be convenient to use two different concepts to obtain sub-frameworks of AFs. Let $F = (A, R)$ be an AF and S a set of arguments. Then, $F|_S = ((A \cap S), R \cap (S \times S))$ is the *sub-framework* of F w.r.t. S and we also use $F - S = F|_{A \setminus S}$. We note the following relation (which we use implicitly later on), for an AF F and sets S, S': $F|_{S \setminus S'} = F|_S - S' = (F - S')|_S$. In particular, for an AF F, a component $C \in SCCs(F)$ and a set S we thus have $F|_{C \setminus S} = F|_C - S$.

For the framework F from Example 5 and the set $S = \{f\}$, $F|_{C_3} - S = (\{e, g, h, i\}, \{(g, h), (h, i)\})$. We now give a definition of *cf2* semantics which only differs in notation from (but is equivalent to) the original definition in [12]. We use some of the notation established above, like the concept of sub-frameworks and the corresponding relations. Moreover,

- $D_F(S)$, the set of *component defeated* arguments, identifies a set of arguments which is defeated by a set S from outside their component, and replaces the set "$D_F(S, E)$";
- $F|_C - D_F(S)$ replaces "$F_{\downarrow UP_F(S,E)}$";
- the set of unattacked arguments "$U_F(S, E)$" as used in the general schema from [12], is not required here, because the base function for the *cf2* semantics does not make use of it.

Definition 12. *Let $F = (A, R)$ be an AF and $S \subseteq A$. An argument $b \in A$ is* component-defeated *by S (in F), if there exists an $a \in S$, such that $(a, b) \in R$ and $a \notin C_F(b)$. The set of arguments component-defeated by S in F is denoted by $D_F(S)$.*

Then, the *cf2* extensions of an AF are recursively defined as follows.

Definition 13. *Let $F = (A, R)$ be an argumentation framework and S a set of arguments. Then, S is a cf2 extension of F, i.e. $S \in cf2(F)$, iff*

- *$S \in naive(F)$, in case $|SCCs(F)| = 1$;*
- *otherwise, $\forall C \in SCCs(F)$, $(S \cap C) \in cf2(F|_C - D_F(S))$.*

In words, the recursive definition $cf2(F)$ is based on a decomposition of the AF F into its $SCCs$ depending on a given set S of arguments. We illustrate the behavior of this procedure in the following example.

Example 6. *Consider the framework F from Example 5. We check whether $S = \{a, d, e, g, i\}$ is a cf2 extension of F (the arguments of the set S are highlighted in Figure 2.5). Following Definition 13, we first identify the SCCs of F, hence $SCCs(F) = \{C_1, C_2, C_3\}$ as in Example 5. Due to the attack (d, f) and $d \in S$ we obtain f as the only component-defeated argument, thus $D_F(S) = \{f\}$. This leads us to the following checks (see also Figure 2.6 which shows the involved sub-frameworks). Note here that in case $F|_{C_i} - D_F(S) = F|_{C_i}$ we only write $(S \cap C_i) \in cf2(F|_{C_i})$.*

1. *$(S \cap C_1) \in cf2(F|_{C_1})$: the sub-framework $F|_{C_1}$ consists of a single SCC; hence, we have to check whether $(S \cap C_1) = \{a\} \in naive(F|_{C_1})$, which indeed holds.*
2. *$(S \cap C_2) \in cf2(F|_{C_2})$: the sub-framework $F|_{C_2}$ consists of a single argument d (and thus of a single SCC); $(S \cap C_2) = \{d\} \in naive(F|_{C_2})$ thus holds.*
3. *$(S \cap C_3) \in cf2(F|_{C_3} - \{f\})$: the sub-framework $F|_{C_3} - \{f\} = F|_{\{e,g,h,i\}}$ consists of four SCCs, namely $C_4 = \{e\}$, $C_5 = \{g\}$, $C_6 = \{h\}$ and $C_7 = \{i\}$. Hence, we need a second level of recursion for $F' = F|_{\{e,g,h,i\}}$ and $S' = S \cap C_3$. Note that we have $D_{F'}(S') = \{h\}$. The single-argument AFs $F'|_{C_4} = F|_{\{e\}}$, $F'|_{C_5} = F|_{\{g\}}$, $F'|_{C_7} = F|_{\{i\}}$ all satisfy $(S' \cap C_i) \in naive(F'|_{C_i})$; while $F'|_{C_6} - \{h\}$ yields the empty AF. Therefore, $(S' \cap C_6) = \emptyset \in cf2(F|_{C_6} - \{h\})$ holds as well.*

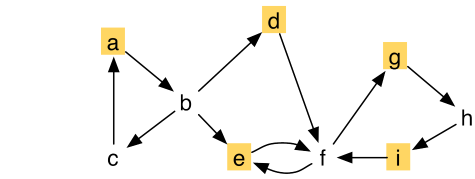

Figure 2.5: The argumentation framework F from Example 5.

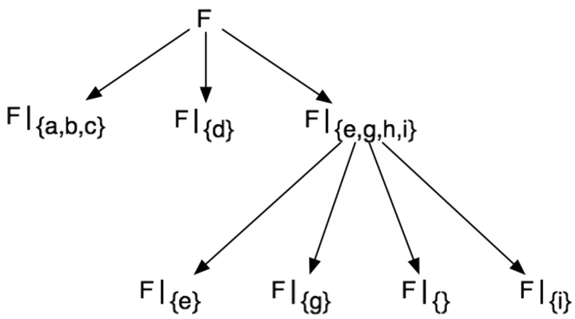

Figure 2.6: Tree of recursive calls for computing $cf2(F)$ from Example 5.

We thus conclude that S is a cf2 extension of F. Further cf2 extensions of F are $\{b, f, h\}$, $\{b, g, i\}$ and $\{c, d, e, g, i\}$. The extensions of the other semantics for this example are as follows:

- $stable(F) = \emptyset$;
- $grd(F) = grd^*(F) = \{\emptyset\}$;
- $adm(F) = compl(F) = \{\emptyset, \{g, i\}\}$;
- $pref(F) = semis(F) = ideal(F) = \{\{g, i\}\}$.

For the stage semantics we obtain the same result as for the cf2 semantics, but this is not the case in general, as we are going to discuss in the next section. ◇

2.2 Properties of the Semantics

In the previous section we already discussed the differences between most of the semantics, especially the basic semantics defined by Dung are very well known. As the focus of this work is mainly on naive-based semantics and out of them the *cf2* semantics, we point out here some special properties and differences between those semantics, where our analysis will be mostly example-driven, and we classify the semantics w.r.t. their subset-relation.

The first example we consider in this context shows one significant difference between *cf2* and stage semantics.

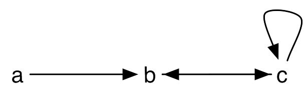

Figure 2.7: The argumentation framework F from Example 7.

Example 7. *Let $F = (A, R)$ with $A = \{a, b, c\}$ and $R = \{(a, b), (b, c), (c, b), (c, c)\}$ as in Figure 2.7. Then, the above defined semantics yield the following extensions.*

- $stable(F) = \emptyset$;
- $adm(F) = \{\{\}, \{a\}\}$;
- $pref(F) = grd(F) = \{\{a\}\}$; *and*
- $naive(F) = \{\{a\}, \{b\}\}$.

Regarding cf2, we check for the two naive sets $S = \{a\}$ and $T = \{b\}$ if they are cf2 extensions of F. As F has two SCCs $C_1 = \{a\}$ and $C_2 = \{b, c\}$, $D_F(S) = \{b\}$ and $D_F(T) = \emptyset$. We first check if $S \in cf2(F)$ as in Definition 13 .

- $(S \cap C_1) \in cf2(F|_{C_1})$ *holds as* $\{a\} \in naive(F|_{C_1})$, *and*
- $(S \cap C_2) \in cf2(F|_{C_2} - \{b\})$ *holds as* $\emptyset \in naive(F|_{\{c\}})$.

Thus, $S \in cf2(F)$. Next we make the check for the set T.

- $(T \cap C_1) \notin cf2(F|_{C_1})$ *because* $(T \cap C_1) = \emptyset$ *and* $naive(F|_{C_1}) = \{\{a\}\}$,
- $(T \cap C_2) \in cf2(F|_{C_2})$ *holds as* $\{b\} \in naive(F|_{C_2})$.

As the first check for T failed, we obtain that $T \notin cf2(F)$.

Regarding stage semantics, both sets S and T are stage extensions. If we have a closer look at the set T, we see that $T_R^+ = \{b, c\}$ and there is no $U \in cf(F)$ s.t. $U_R^+ \supset T_R^+$. $\diamond$

The AF of this example shows that stage semantics can accept an extension which does not include the grounded extension. Moreover, the stage extension $T = \{b\}$ is attacked by the unattacked argument a. This can be seen as a drawback and, besides naive sets, stage semantics is the only one considered so far showing this behavior. In Chapter 4 we introduce the new semantics *stage2* which repairs this drawback.

For stable semantics we already mentioned that it is the only semantics where it can be the case, that there does not exist any extension. This is due to the fact that the requirements for stable semantics are very strong. Furthermore, stable semantics is the only one falling into both categories, the admissible-based and the naive-based semantics.

Next we consider in more detail the *cf2* semantics, as it has some special properties which clearly differ from the admissible-based semantics. Especially the treatment of odd- and even-length cycles is more uniform in the case of *cf2* semantics.

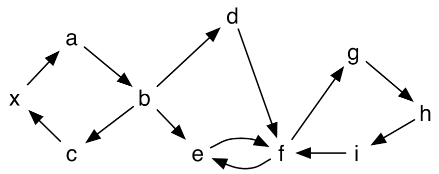

Figure 2.8: The modified AF F' from Example 5.

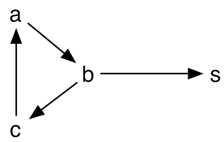

Figure 2.9: Framework F.

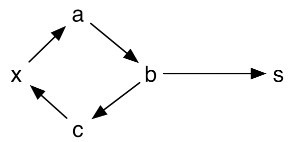

Figure 2.10: Modified Framework G.

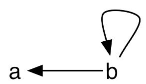

Figure 2.11: AF F from Example 9.

For our framework from Example 5 we obtain $\{g, i\}$ as the only preferred extension. This comes due to the fact that in an odd-length cycle, as we have it in this example none of the arguments a, b and c can be defended. We modify the framework in the sense that we include a new argument x which makes the cycle even, as illustrated in Figure 2.8. Then, we obtain totally different preferred extensions, namely $\{b, x, g, i\}$, $\{b, x, f, h\}$ and $\{a, c, d, e, g, i\}$ which are conform with the *cf2* extensions of the modified AF F'.

The main motivation behind selecting arguments out of an odd-length cycle is to see the arguments as different choices and to be able to choose between them. Then, there is no need for defense. Consider the following example which illustrates this idea [88].

Example 8. *Suppose there are three witnesses A, B and C, where A states that B is unreliable, B states that C is unreliable and C states that A is unreliable. Moreover, C has a statement S. The graph of the framework F is illustrated in Figure 2.9. Any admissible-based semantics returns the empty set as its only extension. But if we have four rather than three witnesses, let's call the fourth one X, as in the AF G pictured in Figure 2.10, the situation changes, and the preferred extensions of G are $\{a, c, s\}$ and $\{b, x\}$. On the other hand, the naive-based semantics return $stage(F) = cf2(F) = \{\{b\}, \{a, s\}, \{c, s\}\}$ and $stage(G) = cf2(G) = \{\{a, c, s\}, \{b, x\}\}$.* $\diamond$

One special case of an odd-length cycle are self-attacking arguments.

Example 9. *Consider the AF F as in Figure 2.11. Then, the empty set is the only preferred extension, whereas $\{a\}$ is a cf2 extension. The motivation behind selecting $\{a\}$ as a reasonable*

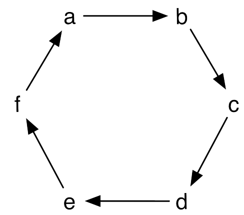

Figure 2.12: AF from Example 10.

extension is that it is not necessary to defend a against the attack from b, as b is a self-attacking argument. $\diamond$

Till now, we only mentioned positive properties of the $cf2$ semantics compared to admissible-based semantics. The next example will show a more questionable behavior.

Example 10. *Consider the AF F in Figure 2.12. We obtain*

- $stage(F) = pref(F) = stable(F) = \{\{a, c, e\}, \{b, d, f\}\}$*, but*
- $cf2(F) = naive(F) = \{\{a, d\}, \{b, e\}, \{c, f\}, \{a, c, e\}, \{b, d, f\}\}$.

In this example we have an even-length cycle and the $cf2$ semantics produce three more extensions. This does not really coincide with the motivation for a symmetric treatment of odd- and even-length cycles, as now the results differ significantly for an even-length cycle. $\diamond$

One suggestion to repair the undesired behavior from Example 10, is to check in Definition 13 for the case $|SCCs(F)| = 1$ whether $S \in stage(F)$ instead of $S \in naive(F)$. In Chapter 4 we formalize this modification of $cf2$ semantics and introduce a new semantics, the $stage2$ semantics [44].

As pointed out in Example 6, there is no particular relation between the $cf2$ and the preferred semantics, but the stage and the $cf2$ semantics coincide for this framework. The following examples will show that in general there is no particular relation between stage and $cf2$ extensions as well.

Example 11. *Consider the AF F in Figure 2.14. Here $\{a, c\}$ is the only stage extension of F (it is also stable). Concerning $cf2$ semantics, note that F is built from a single SCC. Thus, the $cf2$ extensions are given by the naive sets of F, which are $\{a, c\}$ and $\{a, d\}$. Thus, we have $stage(F) \subset cf2(F)$.* $\diamond$

As an example for a framework F such that $cf2(F) \subset stage(F)$, consider the AF from Example 7, where $cf2(F) = \{\{a\}\}$ but $stage(F) = \{\{a\}, \{b\}\}$.

The relations between the introduced semantics are illustrated in Figure 2.13, an arrow from semantics σ to semantics τ encodes that each σ extension is also a τ extension [7, 10, 14, 17, 25, 26, 37, 38, 96].

Finally, we consider a class of frameworks where stable and preferred semantics coincide, the so called coherent AFs [37].

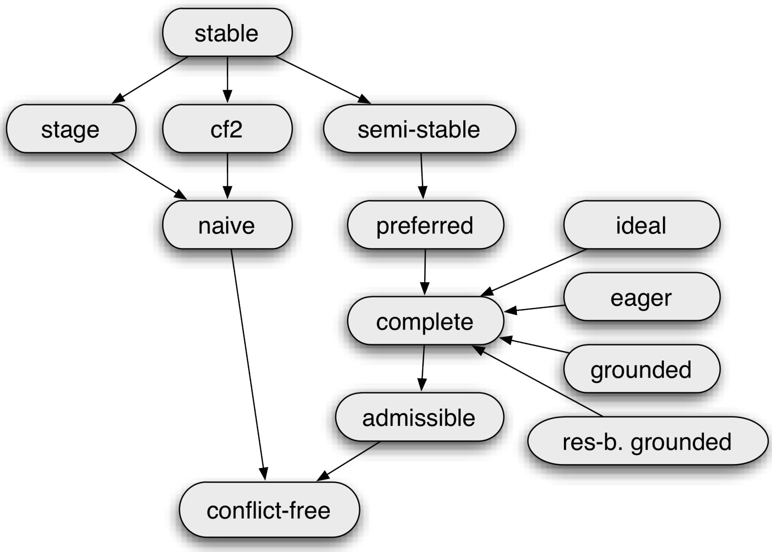

Figure 2.13: Relations between semantics.

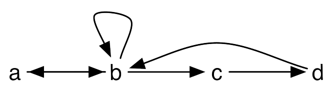

Figure 2.14: AF F from Example 11.

Definition 14. *An AF F is* coherent *if each preferred extension of F is a stable extension of F.*

It follows that coherent AFs are odd-cycle free [37]. Furthermore in coherent AFs also semi-stable and stage semantics coincide with preferred [47]. Whereas this does not hold for $cf2$ semantics as one can see in Example 10. There, F is coherent but $cf2(F) \neq \sigma(F)$, where $\sigma = \{stable, stage, pref, semis\}$.

2.3 Evaluation Criteria

For a long time the analysis of properties of many argumentation semantics was only example driven as shown in the previous section. The advantage of this method is to better understand the behavior of the semantics on different example AFs. Whereas, for a more general understanding and classification of the semantics a systematic analysis is very important. A first step towards this direction was made by Baroni and Giacomin in [8], where they introduced several evaluation criteria for the semantics. In this section we analyze the criteria relevant for naive-based semantics. First we give the definitions for the *extension evaluation criteria*.

Definition 15. *A semantics σ satisfies*

- *the I-*maximality criterion *if for each AF $F = (A, R)$, and for each $S_1, S_2 \in \sigma(F)$, if $S_1 \subseteq S_2$ then $S_1 = S_2$;*

- *the* reinstatement criterion *if for each AF $F = (A, R)$, and for each $S \in \sigma(F)$, if an argument a is defended by S, this implies $a \in S$.*

- *the* weak reinstatement criterion*, if for each $F = (A, R)$, and for each $S \in \sigma(F)$, $grd(F) \subseteq S$;*

- *the $\mathcal{CF}$-*reinstatement criterion*, if for each $F = (A, R)$, for each $S \in \sigma(F), \forall b : (b, a) \in R, \exists c \in S : (c, b) \in R$ and $S \cup \{a\} \in cf(F) \Rightarrow a \in S$.*

- *the* directionality criterion *if for each $F = (A, R)$, and for each set of unattacked arguments $U \subseteq A$ (s.t. $\forall a \in A \setminus U$ there is no $b \in U$ with $(a, b) \in R$), it holds that $\sigma(F|_U) = \{(S \cap U) \mid S \in \sigma(F)\}$.*

The I-maximality criterion states that no extension is a strict subset of another one. All semantics considered here, except complete semantics, satisfy this basic criterion. The reinstatement criterion requires that an argument that is defended by an extension should also belong to the extension. Unsurprisingly, this criterion is not satisfied by stage and *cf2* semantics, as both semantics are not based on the notion of defense.

Therefore, one can consider the weaker forms of this criterion, namely the weak- and $\mathcal{CF}$-reinstatement. The first one claims that the grounded extension should be contained in any extension, whereas the latter requires that if an argument is defended by the extension and is not in conflict with the extension, then it should belong to the extension. For any semantics σ we have the relation, if σ satisfies the reinstatement criterion then it satisfies also the two weaker forms. Furthermore, if σ satisfies weak reinstatement then it satisfies also $\mathcal{CF}$-reinstatement. The other direction does not hold in general. For *cf2* semantics we have the case that weak reinstatement is fulfilled.

Last, the directionality criterion considers that arguments can affect each other only following the direction of attacks. Then, unattacked sets of arguments should be unaffected by the remaining part of the AF [14]. This criterion is not satisfied by stable, stage and semi-stable semantics.

Next we recall the *skepticism related criteria* according to [8, 17]. We start with two skepticism relations between sets of extensions, where $\sigma_1 \preceq^E \sigma_2$ means σ_1 is at least as skeptical as σ_2.

Definition 16. *Let σ_1 and σ_2 be two sets of extensions of an AF F, then*

- *the* elementary skepticism relation *is defined as $\sigma_1 \preceq^E_\cap \sigma_2$ iff*

$$\bigcap_{S_1 \in \sigma_1} S_1 \subseteq \bigcap_{S_2 \in \sigma_2} S_2;$$

	naive	*stable*	*stage*	*cf2*	*grd*	*compl*	*pref*	*semis*	*ideal*	*grd**
I-max.	?	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Reinst.	?	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Weak reinst.	?	Yes	?	Yes	Yes	Yes	Yes	Yes	Yes	Yes
$\mathcal{CF}$-reinst.	?	Yes	?	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Direct.	?	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes
$\preceq^E_\cap$-sk. ad.	?	Yes	?	Yes	Yes	Yes	No	No	No	Yes
$\preceq^E_W$-sk. ad.	?	Yes	?	Yes	Yes	Yes	No	No	No	Yes

Table 2.1: Evaluation criteria w.r.t. the introduced semantics.

- *the* weak skepticism relation *is defined as* $\sigma_1 \preceq^E_W \sigma_2$ *iff*

$$\forall S_2 \in \sigma_2 \exists S_1 \in \sigma_1 : S_1 \subseteq S_2.$$

In [8] there was also defined the strong skepticism relation $\preceq^E_S$ but as stated in [17], this relation is too strong as it prevents to compare any pair of multiple-status semantics. Therefore, we will not consider the strong skepticism relation in this work.

To compare semantics w.r.t. the above defined skepticism relations we also need to be able to compare AFs. The following definition states when to AFs are *comparable*.

Definition 17. *Let $F = (A, R_1)$ and $G = (A, R_2)$, $F \preceq^A G$ iff $conf(F) = conf(G)$ and $R_2 \subseteq R_1$. Where $conf(F) = \{(a, b) \in R \mid (a, b) \vee (b, a) \in R\}$ is the set of* conflicting pairs of arguments.

Skepticism adequacy is now granted for a semantics σ if for any two comparable AFs the skepticism relation between their sets of σ extensions is preserved.

Definition 18. *Given a skepticism relation $\preceq^E$ according to Definition 16, a semantics σ is* $\preceq^E$-skepticism adequate *iff for any AFs F, G such that $F \preceq^A G$, $\sigma(F) \preceq^E \sigma(G)$ holds.*

The skepticism adequacy properties are ordered in the way that for any semantics σ it holds that if σ satisfies $\preceq^E_W$-skepticism adequacy then, σ satisfies $\preceq^E_\cap$-skepticism adequacy. So clearly if a semantics does not satisfy elementary skepticism than it can not satisfy the stronger form.

In Table 2.1 we summarize the results from [8, 14] for the mentioned evaluation criteria and the introduced semantics[4]. The missing entries for naive and stage semantics will be added in Chapter 4 as they are not included in [8].

[4]We omit here the eager semantics, as it has not been studied in [8, 14]

CHAPTER 3

Alternative Characterization

In the previous section we already discussed the $cf2$ semantics in detail. In particular we pointed out the advantages of this semantics compared to admissible-based semantics. Although these issues were known for some time, the $cf2$ semantics was somehow neglected in the literature. One reason for this might be the recursive definition and the recursive computation of sub-frameworks during the evaluation.

In the original definition of $cf2$ semantics in [12], the computation is based on checking recursively whether a set of arguments fulfills a base function (is a naive set) in a single SCC. Thus, the computation is based on a decomposition of the framework along its SCCs. As normal for a recursive definition, the decomposition is based on the outcome of the base function of the previous step. For an implementation of the algorithm in a standard programming language like JAVA or C++, this definition does not cause any problems and can be encoded straight forward. Whereas designing compact encodings in a declarative way based on this recursive definition is not that easy and one can end up with a quite complicated and difficult to understand encoding. We will come back to this point in Chapter 7.

To facilitate further investigation steps like complexity analysis, analysis of different notions of equivalence and of course the implementation aspects, we introduce an alternative characterization based on the idea to decompose the framework as well, but differently to the original approach the decomposition is only recursive in terms of a certain set of arguments, for which we provide a fixed-point operator. This modification allows us to avoid the recursive computation of several sub-frameworks. Instead we only compute one, possibly not connected, framework where we eliminate the arguments and corresponding attacks which are, what we call, "recursively component defeated". This chapter is organized as follows.

- In Section 3.1 we introduce some formal concepts which we need for the alternative characterization as well as for the correctness proofs.
- In Section 3.2 we successively introduce the alternative characterization, where we first use the set $\mathcal{RD}_F(S)$, the recursively component defeated arguments. Then we define the $\Delta_{F,S}$-operator and prove that for a conflict-free set S, $\mathcal{RD}_F(S)$ equals $\Delta_{F,S}$. Then, we

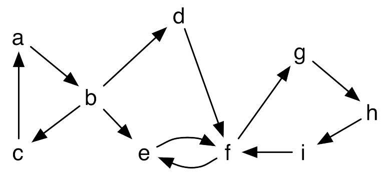

Figure 3.1: Framework F from Example 5.

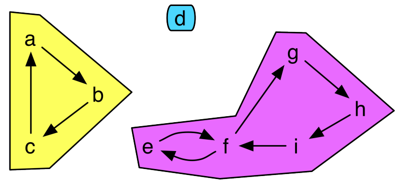

Figure 3.2: Separation of the AF F.

come to our main theorem, an alternative characterization of *cf2* which does not require a recursive computation of several sub-frameworks.

- Finally in Section 3.3 we close the chapter with an analysis.

Parts of this work have been published in [67, 69].

3.1 Preliminaries

For the alternative characterization we need some formal concepts which we introduce here. As the *cf2* semantics is based on a decomposition of the framework along its SCCs, the following concepts help to relate graph-theoretic to argumentation-based principles. We start with the *separation*, where an AF is separated if there are no attacks between different strongly connected components.

Definition 19. *An AF $F = (A, R)$ is called* separated *if for each $(a, b) \in R$, $C_F(a) = C_F(b)$. We define the* separation *of F as*

$$[[F]] = \bigcup_{C \in SCCs(F)} F|_C.$$

The separation of an AF always yields a separated AF. For example the separation of the framework F from Example 5 is depicted in Figure 3.2. For comparison, the attacks (b, d), (b, e) and (d, f) of the original framework as shown in Figure 3.1 were eliminated, as they are situated between the different SCCs $C_1 = \{a, b, c\}$, $C_2 = \{d\}$ and $C_3 = \{e, f, g, h, i\}$.

The following technical lemma will be useful later.

Lemma 2. *For any AF F and set S of arguments,*

$$\bigcup_{C \in SCCs(F)} [[F|_C - S]] = [[F - S]].$$

Proof. We first note that for disjoint AFs F and G, $[[F]] \cup [[G]] = [[F \cup G]]$ holds. Moreover, for a set S of arguments and arbitrary frameworks F and G, $(F - S) \cup (G - S) = (F \cup G) - S$ is clear. Using these observations, we obtain

$$\bigcup_{C \in SCCs(F)} [[F|_C - S]] = [[\bigcup_{C \in SCCs(F)} (F|_C - S)]] = [[(\bigcup_{C \in SCCs(F)} F|_C) - S]] = [[[[F]] - S]].$$

It remains to show that $[[[[F]]-S]] = [[F-S]]$. Obviously, both AFs possess the same arguments A. Thus, let R be the attacks of $[[[[F]] - S]]$ and R' the attacks of $[[F - S]]$. $R \subseteq R'$ holds by the fact that each attack in $[[F]]$ is also contained in F. To show $R' \subseteq R$, let $(a, b) \in R'$. Then $a, b \notin S$, and $C_{F-S}(a) = C_{F-S}(b)$. From the latter, $C_F(a) = C_F(b)$ and thus (a, b) is an attack in $[[F]]$ and also in $[[F]] - S$. Again using $C_{F-S}(a) = C_{F-S}(b)$, shows $(a, b) \in R$. $\square$

Next, we define $\ell_F(S)$, the level of recursiveness a framework shows with respect to a set S of arguments and then the aforementioned set $\mathcal{RD}_F(S)$, the set of recursively component defeated arguments (by S) in an AF F.

Definition 20. *For an AF $F = (A, R)$ and a set S of arguments, we recursively define the* level *$\ell_F(S)$ of F w.r.t. S as follows:*

- *if $|SCCs(F)| = 1$ then $\ell_F(S) = 1$;*
- *otherwise, $\ell_F(S) = 1 + max(\{\ell_{F|_{C-D_F(S)}}(S \cap C) \mid C \in SCCs(F)\})$.*

For the AF F from Example 5 (Figure 3.1) we obtain the level $\ell_F(S)$ w.r.t. the set $S = \{a, d, e, g, i\}$ as follows. $\ell_F(S) = 1+max(\{\ell_{F|_{C-D_F(S)}}(S\cap C) \mid C \in SCCs(F)\})$, where $D_F(S) = \{f\}$ and $SCCs(F) = \{C_1, C_2, C_3\}$ with $C_1 = \{a, b, c\}$, $C_2 = \{d\}$ and $C_3 = \{e, f, g, h, i\}$. This leads to the following recursive calls:

- $\ell_{F|_{C_1}}(S \cap C_1) = 1$,
- $\ell_{F|_{C_2}}(S \cap C_2) = 1$,
- $\ell_{F'}(S') = 1 + max(\{\ell_{F'|_{C'-D_{F'}(S')}}(S' \cap C') \mid C' \in SCCs(F')\})$. Where $F' = F|_{C_3} - D_F(S)$, $S' = S \cap C_3 = \{e, g, i\}$ and $D_{F'}(S') = \{h\}$, furthermore $SCCs(F') = \{C_4, C_5, C_6, C_7\}$ with $C_4 = \{e\}$, $C_5 = \{g\}$, $C_6 = \{h\}$ and $C_7 = \{i\}$. As all those SCCs of F' are single SCCs, we obtain in each recursive call level 1.

To sum up, the level of F w.r.t. S is $\ell_F(S) = 3$. One can compare the tree of recursive calls in Figure 2.5 with the computation of $\ell_F(S)$. When the *height* h of a tree is the length of the path from the root to the deepest node in the tree, we denote the height of the computation tree for the *cf2* semantics for an AF F w.r.t. S as $h_F(S)$, then $\ell_F(S) = h_F(S) + 1$.

The next definition is very important for the alternative characterization as it allows us to recursively compute the component defeated arguments. Remember, in Definition 12 we defined $D_F(S)$, the set of component defeated arguments which only gives us the "locally" component defeated arguments. Here we want to compute recursively those arguments, attacked by a set S, where in each recursive call the current evaluation has an influence on the next call. In particular the SCCs of the sub-frameworks may change.

Definition 21. *Let $F = (A, R)$ be an AF and S a set of arguments. We define $\mathcal{RD}_F(S)$, the set of arguments* recursively component defeated *by S (in F) as follows:*

- *if $|SCCs(F)| = 1$ then $\mathcal{RD}_F(S) = \emptyset$;*
- *otherwise, $\mathcal{RD}_F(S) = D_F(S) \cup \bigcup_{C \in SCCs(F)} \mathcal{RD}_{F|_{C-D_F(S)}}(S \cap C)$.*

Consider the AF F from Example 5 (Figure 3.1), and the set $S = \{a, d, e, g, i\}$. The SCCs of F are $C_1 = \{a, b, c\}$, $C_2 = \{d\}$ and $C_3 = \{e, f, g, h, i\}$ and $D_F(S) = \{f\}$. Then, following Definition 21, the set of recursively component defeated arguments are computed as follows, $\mathcal{RD}_F(S) = \{f\} \cup \bigcup_{C \in SCCs(F)} \mathcal{RD}_{F|_C - \{f\}}(S \cap C)$ where the next recursive calls are:

- $\mathcal{RD}_{F|_{C_1}}(\{a\}) = \emptyset$,
- $\mathcal{RD}_{F|_{C_2}}(\{d\}) = \emptyset$,
- $\mathcal{RD}_{F|_{\{e,g,h,i\}}}(\{e, g, i\}) = \{h\} \cup \bigcup_{C \in SCCs(F|_{\{e,g,h,i\}})} \mathcal{RD}_{F|_C - \{h\}}(\{e, g, i\} \cap C)$.

The last calls all lead to empty sets as the sub-frameworks all consist of single SCCs or are empty in the case of $F|_{\{h\}} - \{h\}$. Thus, we finally obtain $\mathcal{RD}_F(S) = \{f, h\}$.

3.2 New Characterization for *cf2* Semantics

We are now prepared to give our first alternative characterization, which establishes a *cf2* extension S of a given AF F by checking whether S is a naive extension of a certain separated framework constructed from F using S.

Lemma 3. *Let $F = (A, R)$ be an AF and S be a set of arguments. Then,*

$$S \in \mathit{cf2}(F) \text{ iff } S \in \mathit{naive}([[F - \mathcal{RD}_F(S)]]).$$

Proof. We show the claim by induction over $\ell_F(S)$.

Induction base. For $\ell_F(S) = 1$, we have $|SCCs(F)| = 1$. By definition, $\mathcal{RD}_F(S) = \emptyset$ and we have $[[F - \mathcal{RD}_F(S)]] = [[F]] = F$. Thus, the assertion states that $S \in \mathit{cf2}(F)$ iff $S \in \mathit{naive}(F)$ which matches the original definition for *cf2* semantics in case F consists of a single strongly connected component.

Induction step. Let $\ell_F(S) = n$ and assume the assertion holds for all AFs F' and sets S' with $\ell_{F'}(S') < n$. In particular, by Definition 20, for each $C \in SCCs(F)$ we have $\ell_{F|_C - D_F(S)}(S \cap C) < n$. By the induction hypothesis, we thus obtain that for each $C \in SCCs(F)$ the following holds:

$$(S \cap C) \in \mathit{cf2}\big(F|_C - D_F(S)\big) \text{ iff } (S \cap C) \in \mathit{naive}\Big([[\big(F|_C - D_F(S)\big) - \mathcal{R}'_{F,C,S}]]\Big) \quad (3.1)$$

where $\mathcal{R}'_{F,C,S} = \mathcal{RD}_{F|_C - D_F(S)}(S \cap C)$. Let us fix a $C \in SCCs(F)$.

$$\big(F|_C - D_F(S)\big) - \mathcal{R}'_{F,C,S} = \quad (3.2)$$

$$\Big(\big(F|_C - D_F(S)\big) - \mathcal{R}'_{F,C,S}\Big) - \bigcup_{C' \in SCCs(F); C \neq C'} \mathcal{RD}_{F|_{C'} - D_F(S)}(S \cap C') = \quad (3.3)$$

$$\big(F|_C - D_F(S)\big) - \bigcup_{C \in SCCs(F)} \mathcal{RD}_{F|_C - D_F(S)}(S \cap C) = \quad (3.4)$$

$$F|_C - \Big(D_F(S) \cup \bigcup_{C \in SCCs(F)} \mathcal{RD}_{F|_C - D_F(S)}(S \cap C)\Big) = F|_C - \mathcal{RD}_F(S). \quad (3.5)$$

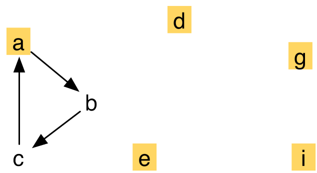

Figure 3.3: The separation $[[F - \mathcal{RD}_F(S)]]$ from Example 12.

As we fixed a $C \in SCCs(F)$ we come from (3.2) to (3.3) because for each further $C' \in SCCs(F)$ (i.e. $C \neq C'$), no argument from $\mathcal{RD}_{F|_{C'}-D_F(S)}(S \cap C')$ occurs in $F|_C$. From (3.3) to (3.4): $\mathcal{R}'_{F,C,S}$ is defined for the SCC C and $\bigcup_{C' \in SCCs(F); C \neq C'} \mathcal{RD}_{F|_{C'}-D_F(S)}(S \cap C')$ for all other SCCs $C \neq C'$, then in (3.4) we put all these SCCs together. In (3.5), by Definition (21) we obtain in the brackets $\mathcal{RD}_F(S)$.

Thus, for any $C \in SCCs(F)$, relation (3.1) amounts to

$$(S \cap C) \in \mathit{cf2}\big(F|_C - D_F(S)\big) \text{ iff } (S \cap C) \in \mathit{naive}\big([[F|_C - \mathcal{RD}_F(S)]]\big). \tag{3.6}$$

We now prove the assertion. Let $S \in \mathit{cf2}(F)$. By Definition 13, for each $C \in SCCs(F)$, $(S \cap C) \in \mathit{cf2}(F|_C - D_F(S))$. We use (3.6), then for each $C \in SCCs(F)$ it follows that $(S \cap C) \in \mathit{naive}([[F|_C - \mathcal{RD}_F(S)]])$. By the definition of components and the semantics of being naive, the following relation follows:

$$\bigcup_{C \in SCCs(F)} (S \cap C) \in \mathit{naive}\Big(\bigcup_{C \in SCCs(F)} [[F|_C - \mathcal{RD}_F(S)]]\Big).$$

Since $S = \bigcup_{C \in SCCs(F)}(S \cap C)$ and, by Lemma 2, $\bigcup_{C \in SCCs(F)}[[F|_C - \mathcal{RD}_F(S)]] = [[F - \mathcal{RD}_F(S)]]$, we arrive at $S \in \mathit{naive}([[F - \mathcal{RD}_F(S)]])$ as desired. The other direction is by essentially the same arguments. □

The definition of *cf2* from Lemma 3 allows us to make only one check for each possible set S in one sub-framework. We consider another time the AF of Example 5 (Figure 3.1).

Example 12. *Let $F = (A, R)$ from Example 5 (Figure 3.1) and $S = \{a, d, e, g, i\}$. Then $\mathcal{RD}_F(S) = \{f, h\}$ and the separation $[[F - \mathcal{RD}_F(S)]]$ is depicted in Figure 3.3 where the arguments in S are highlighted. It is easy to see that S is a naive extension of the separation of F w.r.t. $\mathcal{RD}_F(S)$.* ◇

Note, the set of recursively component defeated arguments can be different for each set $S \subseteq A$ and therefore, also the separation may vary. The main difference of the characterization in Lemma 3 to the one in Definition 13 is that the recursion has been shifted to $\mathcal{RD}_F(S)$, and there is only one check for a set S to be a naive extension of a sub-framework of F.

We can not get rid of the recursion in the definition of *cf2* but, the computation of several sub-frameworks, which is still the case in the computation of $\mathcal{RD}_F(S)$, can be avoided, as we will show next when we introduce the $\Delta_{F,S}$-operator.

$\Delta_{F,S}$-Operator

In this subsection, we provide an alternative characterization for $\mathcal{RD}_F(S)$ via a fixed-point operator. In other words, this yields a linearization in the recursive computation of this set. To this end, we require a parametrized notion of reachability.

Definition 22. *Let $F = (A, R)$ be an AF, B a set of arguments, and $a, b \in A$. We say that b is* reachable *in F from a* modulo B*, in symbols $a \Rightarrow_F^B b$, if there exists a path from a to b in $F|_B$, i.e. there exists a sequence $c_1, \ldots, c_n$ ($n > 1$) of arguments such that $c_1 = a$, $c_n = b$, and $(c_i, c_{i+1}) \in R \cap (B \times B)$, for all i with $1 \leq i < n$.*

With the reachability at hand we give the definition of the $\Delta_{F,S}$-operator.

Definition 23. *For any AF $F = (A, R)$, $D \subseteq A$, and a set S of arguments,*

$$\Delta_{F,S}(D) = \{a \in A \mid \exists b \in S : b \neq a, (b, a) \in R, a \not\Rightarrow_F^{A \setminus D} b\}.$$

The operator is clearly monotonic, i.e. $\Delta_{F,S}(D) \subseteq \Delta_{F,S}(D')$ holds for $D \subseteq D'$. As usual, we let $\Delta_{F,S}^0 = \Delta_{F,S}(\emptyset)$ and, for $i > 0$, $\Delta_{F,S}^i = \Delta(\Delta_{F,S}^{i-1})$. Due to monotonicity the least fixed-point (lfp) of the operator exists and, with slightly abuse of notation, will be denoted as $\Delta_{F,S}$.

We have a look at our running Example. The AF F from Example 5 (Figure 3.1) and the set $S = \{a, d, e, g, i\}$, then in the first iteration of computing the least fixed-point of $\Delta_{F,S}$, we have $\Delta_{F,S}(\emptyset) = \{f\}$ because the argument f is the only one which is attacked by S but its attacker d is not reachable by f in F. In the second iteration, we obtain $\Delta_{F,S}(\{f\}) = \{f, h\}$ because h is attacked by $g \in S$ and h can not reach its attacker in the framework $F - \{f\}$. Finally, in the third iteration we reach the least fixed-point with $\Delta_{F,S}(\{f, h\}) = \{f, h\}$.

We need two more lemmata before showing that $\Delta_{F,S}$ captures $\mathcal{RD}_F(S)$. The first one states that $\Delta_{F,S}^0$ computes the (locally) component defeated arguments.

Lemma 4. *For any AF $F = (A, R)$ and any set $S \subseteq A$, $\Delta_{F,S}^0 = D_F(S)$.*

Proof. We have $\Delta_{F,S}^0 = \Delta_{F,S}(\emptyset) = \{a \in A \mid \exists b \in S : b \neq a, (b, a) \in R, a \not\Rightarrow_F^A b\}$. Hence, $a \in \Delta_{F,S}^0$, if there exists a $b \in S$, such that $(b, a) \in R$ and a does not reach b in F, i.e. $b \notin C_F(a)$. This meets exactly the definition of $D_F(S)$. □

We next prove a certain property $\Delta_{F,S}$ satisfies w.r.t. the components of F.

Lemma 5. *For any AF $F = (A, R)$ and any set $S \in cf(F)$,*

$$\Delta_{F,S} = D_F(S) \cup \bigcup_{C \in SCCs(F)} \Delta_{F|_C - D_F(S), (S \cap C)}.$$

Proof. Let $F = (A, R)$. For the $\subseteq$-direction, we show by induction over $i \geq 0$ that

$$\Delta_{F,S}^i \subseteq D_F(S) \cup \bigcup_{C \in SCCs(F)} \Delta_{F|_C - D_F(S), (S \cap C)}.$$

To ease notation, we write $\bar{\Delta}_{F,S,C}$ as a shorthand for $\Delta_{F|_C - D_F(S),(S\cap C)}$, where $C \in SCCs(F)$.

Induction base. For $i = 0$, $\Delta^0_{F,S} \subseteq D_F(S) \cup \bigcup_{C\in SCCs(F)} \bar{\Delta}_{F,S,C}$ follows from Lemma 4.

Induction step. Let $i > 0$ and assume $\Delta^j_{F,S} \subseteq D_F(S) \cup \bigcup_{C\in SCCs(F)} \bar{\Delta}_{F,S,C}$ holds for all $j < i$. Let $a \in \Delta^i_{F,S}$. Then, there exists a $b \in S$, such that $(b, a) \in R$ and $a \not\Rightarrow^D_F b$, where $D = A \setminus \Delta^{i-1}_{F,S}$. If $b \notin C_F(a)$, we have also $a \not\Rightarrow^A_F b$ and thus $a \in D_F(S)$. Hence, suppose $b \in C_F(a)$. Then, $a \notin D_F(S)$ and, since $S \in cf(F)$ and $b \in S$, also $b \notin D_F(S)$. Thus, both a and b are contained in the framework $F|_C - D_F(S)$ (and so is the attack (b, a)) for $C = C_F(a)$. Moreover, $b \in (S \cap C)$. Towards a contradiction, assume now $a \notin \bar{\Delta}_{F,S,C}$. This yields that $a \Rightarrow^{D'}_{F|_C - D_F(S)} b$ for $D' = A \setminus \bar{\Delta}_{F,S,C}$, i.e. there exist arguments $c_1, \ldots, c_n$ $(n > 1)$ in $F|_C - D_F(S)$ but not contained in $\bar{\Delta}_{F,S,C}$, such that $c_1 = a$, $c_n = b$, and $(c_i, c_{i+1}) \in R$, for all i with $1 \leq i < n$. Obviously all the c_i's are contained in F as well, but since $a \not\Rightarrow^D_F b$ (recall that $D = A \setminus \Delta^{i-1}_{F,S}$), it must hold that at least one of the c_i's, say c, has to be contained in $\Delta^{i-1}_{F,S}$. By the induction hypothesis, we get $c \in \bar{\Delta}_{F,S,C}$, a contradiction.

For the $\supseteq$-direction of the claim we proceed as follows. By Lemma 4, we know that $D_F(S) = \Delta^0_{F,S}$ and thus $D_F(S) \subseteq \Delta_{F,S}$. It remains to show that $\bigcup_{C\in SCCs(F)} \Delta_{F|_C - D_F(S),(S\cap C)} \subseteq \Delta_{F,S}$. We show by induction over i that $\Delta^i_{F|_C - D_F(S),(S\cap C)} \subseteq \Delta_{F,S}$ holds for each $C \in SCCs(F)$. Thus, let us fix a $C \in SCCs(F)$ and use $\bar{\Delta}^i_{F,S,C}$ as shorthand for $\Delta^i_{F|_C - D_F(S),(S\cap C)}$.

Induction base. Let $a \in \bar{\Delta}^0_{F,S,C}$. Then, there is a $b \in (S \cap C)$, such that b attacks a in $F' = F|_C - D_F(S)$ and $a \not\Rightarrow^{A'}_{F'} b$, where A' denotes the arguments of F', i.e. $A' = C \setminus D_F(S)$. Since $F|_C$ is built from a SCC C of F, it follows that $a \not\Rightarrow^{A\setminus D_F(S)}_F b$. Since $b \in S$, $(b, a) \in R$, and $D_F(S) = \Delta^0_{F,S}$ (Lemma 4), we get $a \in \Delta^1_{F,S} \subseteq \Delta_{F,S}$.

Induction step. Let $i > 0$ and assume $\bar{\Delta}^j_{F,S,C} \subseteq \Delta_{F,S}$ for all $j < i$. Let $a \in \bar{\Delta}^i_{F,S,C}$. Then, there is a $b \in (S \cap C)$, such that b attacks a in F' and $a \not\Rightarrow^{D'}_{F'} b$, where $D' = A' \setminus \bar{\Delta}^{i-1}_{F,S,C}$. Towards a contradiction, suppose $a \notin \Delta_{F,S}$. Since $b \in S$ and $(b, a) \in R$, it follows that there exist arguments $c_1, \ldots, c_n$ $(n > 1)$ in $F \setminus \Delta_{F,S}$, such that $c_1 = a$, $c_n = b$, and $(c_i, c_{i+1}) \in R$, for all i with $1 \leq i < n$. All these c_i's are thus contained in the same component as a, and moreover these c_i's cannot be contained in $D_F(S)$, since $D_F(S) \subseteq \Delta_{F,S}$. Thus, they are contained in $F|_C - D_F(S)$, but since $a \not\Rightarrow^{D'}_{F'} b$, there is at least one such c_i, say c, contained in $\bar{\Delta}^{i-1}_{F,S,C}$. By the induction hypothesis, $c \in \Delta_{F,S}$, a contradiction. $\square$

Now we are able to obtain the desired relation.

Lemma 6. *For any AF $F = (A, R)$ and any set $S \in cf(F)$, $\Delta_{F,S} = \mathcal{RD}_F(S)$.*

Proof. The proof is by induction over $\ell_F(S)$.

Induction base. For $\ell_F(S) = 1$, $|SCCs(F)| = 1$ by Definition 20. From this and Definition 21, we obtain $\mathcal{RD}_F(S) = D_F(S) = \emptyset$. By Lemma 4, $\Delta^0_{F,S} = D_F(S) = \emptyset$. By definition, $\Delta_{F,S} = \emptyset$ follows from $\Delta^0_{F,S} = \emptyset$.

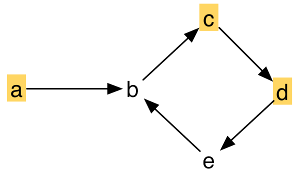

Figure 3.4: Graph from Example 13.

Induction step. Let $\ell_F(S) = n$ and assume the claim holds for all pairs F', $S' \in cf(F')$, such that $\ell_{F'}(S') < n$. In particular, this holds for $F' = F|_C - D_F(S)$ and $S' = (S \cap C)$, with $C \in SCCs(F)$. Note that $(S \cap C)$ is indeed conflict-free in $F|_C - D_F(S)$. By definition we have, $\mathcal{RD}_F(S) = D_F(S) \cup \bigcup_{C \in SCCs(F)} \mathcal{RD}_{F|_C - D_F(S)}(S \cap C)$ and by Lemma 5 we know that $\Delta_{F,S} = D_F(S) \cup \bigcup_{C \in SCCs(F)} \Delta_{F|_C - D_F(S), S \cap C}$. Using the induction hypothesis, i.e. $\Delta_{F|_C - D_F(S), S \cap C} = \mathcal{RD}_{F|_C - D_F(S)}(S \cap C)$, the assertion follows. $\square$

One important part of Lemma 6 is that S needs to be conflict-free in F. In the following example we show that for a set $T \notin cf(F)$, $\Delta_{F,T}$ does not equal $\mathcal{RD}_F(T)$.

Example 13. *Consider the AF $F = (A, R)$ of Figure 3.4, where the set $T = \{a, c, d\}$ is highlighted in the graph. F has two SCCs, namely $C_1 = \{a\}$ and $C_2 = \{b, c, d, e\}$. $\mathcal{RD}_F(T)$ is computed as follows, $\mathcal{RD}_F(T) = \{b\} \cup \bigcup_{C \in SCCs(F)} \mathcal{RD}_{F|_C - \{b\}}(T \cap C)$ where:*

- $\mathcal{RD}_{F|_{C_1}}(\{a\}) = \emptyset$,
- $\mathcal{RD}_{F|_{\{c,d,e\}}}(\{c, d\}) = \{d\} \cup \bigcup_{C \in SCCs(F|_{\{c,d,e\}})} \mathcal{RD}_{F|_C - \{d\}}(\{c, d\} \cap C)$.

The final calls for the SCCs $C_3 = \{c\}$, $C_4 = \{d\}$ and $C_5 = \{e\}$ result in empty sets. Thus, $\mathcal{RD}_F(T) = \{b, d\}$. For comparison we now compute $\Delta_{F,T}$.

- $\Delta_{F,T}(\emptyset) = \{b\}$,
- $\Delta_{F,T}(\{b\}) = \{b, d, e\}$,
- $\Delta_{F,T}(\{b, d, e\}) = \{b, d, e\}$.

Hence, $\Delta_{F,T} = \{b, d, e\} \neq \mathcal{RD}_F(T) = \{b, d\}$. $\diamond$

Main Theorem

We finally reached our main result in this chapter, i.e. an alternative characterization for *cf2* semantics, where the need for recursion is delegated to a fixed-point operator.

Theorem 1. *For any AF F, $cf2(F) = \{S \mid S \in naive(F) \cap naive([[F - \Delta_{F,S}]])\}$.*

Proof. The result holds by the following observations. By Lemma 3, $S \in cf2(F)$ iff $S \in naive([[F - \mathcal{RD}_F(S)]])$. Moreover, from Lemma 6, for any $S \in cf(F)$, $\Delta_{F,S} = \mathcal{RD}_F(S)$. Finally, $S \in cf2(F)$ implies $S \in naive(F)$ (see [14], Proposition 18). $\square$

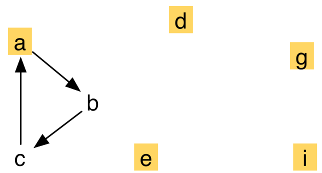

Figure 3.5: Graph of instance $[[F - \Delta_{F,S}]]$ from Example 14.

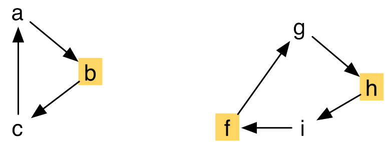

Figure 3.6: Graph of instance $[[F - \Delta_{F,S'}]]$ from Example 14.

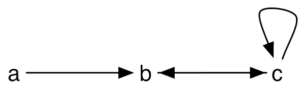

Figure 3.7: Framework F from Example 15.

Figure 3.8: Graph of instance $[[F - \Delta_{F,S}]]$.

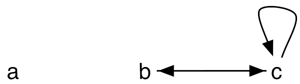

Figure 3.9: Graph of instance $[[F - \Delta_{F,T}]]$.

To illustrate the behavior of the new characterization let us have a look at the following two examples.

Example 14. *Consider the AF F and $S = \{a, d, e, g, i\}$ from Example 5 (Figure 3.1). We already computed $\Delta_{F,S}(\{f\}) = \{f, h\}$ above. Then, $[[F - \Delta_{F,S}]]$ of the AF F w.r.t. S is given by*

$$[[F - \Delta_{F,S}]] = \big(\{a, b, c, d, e, g, i\}, \{(a, b), (b, c), (c, a)\}\big).$$

Figure 3.5 shows the graph of $[[F - \Delta_{F,S}]]$. It is easy to see that $S \in naive([[F - \Delta_{F,S}]])$ as expected, since $S \in cf2(F)$.

For comparison, if we take another set $S' = \{b, f, h\}$, then $\Delta_{F,S'} = \{d, e\}$ and the corresponding instance $[[F - \Delta_{F,S'}]]$ is depicted in Figure 3.6. Also in this case $S' \in cf2(F)$ as $S' \in naive(F)$ and $S' \in naive([[F - \Delta_{F,S'}]])$. $\diamond$

In the next example we illustrate what happens if we apply Theorem 1 to a set $T \notin cf2(F)$.

Example 15. *Let us consider the AF F from Example 7 (Figure 3.7). F has two naive sets, namely $S = \{a\}$ and $T = \{b\}$. First, we concentrate on the set S and compute $\Delta_{F,S} = \{b\}$ and $[[F - \Delta_{F,S}]] = (\{a, c\}, \{(c, c)\})$. Thus, $S \in naive([[F - \Delta_{F,S}]])$ and clearly $S \in cf2(F)$, compare Figure 3.8.*
For T we obtain $\Delta_{F,T} = \emptyset$ and $[[F - \Delta_{F,T}]] = (A, \{(b, c), (c, b), (c, c)\})$ as shown in Figure 3.9. Now, $T \notin naive([[F - \Delta_{F,T}]])$, as there is the set $T' = \{a, b\} \supset T$ and $T' \in naive([[F - \Delta_{F,T}]])$. Thus, $T \notin cf2(F)$. $\diamond$

3.3 Analysis of the New Characterization

With Theorem 1 we gave an alternative characterization for the $cf2$ semantics which does not require a recursive computation of several sub-frameworks. Instead, we shifted the recursion to the computation of the $\Delta_{F,S}$-operator and for a set $S \in naive(F)$ we only compute once a sub-framework, where we delete the arguments in $\Delta_{F,S}$ which are recursively component defeated by S, and in the remaining framework we eliminate attacks between different SCCs. Then, S needs to be a naive extension of the obtained instance of F.

Here we want to say some words about the additional check for a set S to be a naive extension of F. This is not required in Lemma 3, because the definition of $\mathcal{RD}_F(S)$ ensures that the arguments component-defeated by S are not in conflict with each other. To be more precise, each recursive call of $\mathcal{RD}_{F|_{C-D_F(S)}}(S \cap C)$ is responsible for this because in the sub-framework $F|_{C-D_F(S)}$ the arguments in conflict with the component defeated arguments are eliminated. This works similar to the original definition of *cf2* by Baroni et al., where the SCC-recursive schema guarantees that the obtained extensions are conflict-free, if the base function is conflict-free (compare [12], Proposition 47). On the other side, in Theorem 1 we explicitly check if $S \in naive(F)$. For Theorem 1 to be correct, it would be sufficient to check if S is conflict-free in F, but as it is known that each *cf2* extension is also a naive extension, we apply the stronger check. This avoids the computation of the instance $[[F - \Delta_{F,S}]]$ for sets S which are no candidates for *cf2* extension. Whereas, without the requirement $S \in cf(F)$, by Lemma 6 $\Delta_{F,S} \neq \mathcal{RD}_F(S)$ and also $[[F - \Delta_{F,S}]] \neq [[F - \mathcal{RD}_F(S)]]$. Although, we would obtain the *cf2* extensions also in this case, the way how we obtained them is not the same as in the original definition. To exemplify this, let us consider the following example.

Example 16. *Let $F = (A, R)$ from Figure 3.4 and $T = \{a, c, d\}$. $T \notin cf(F)$ and as we know from Example 13, $\Delta_{F,T} = \{b, d, e\} \neq \mathcal{RD}_F(T) = \{b, d\}$. Then, the corresponding instances of F are: $[[F - \Delta_{F,T}]] = (\{a, c\}, \emptyset)$ and $[[F - \mathcal{RD}_F(T)]] = (\{a, c, e\}, \emptyset)$. It is easy to see that T can not be a naive extension of the two instances. As T is not conflict-free there is an argument $d \in T$ such that $d \in \Delta_{F,T}$ and $d \in \mathcal{RD}_F(T)$, hence d is not contained in the two instances and so $T \notin naive([[F - \Delta_{F,T}]])$ and $T \notin naive([[F - \mathcal{RD}_F(T)]])$.* $\diamond$

Still the computation of *cf2* extensions requires some technical notation, but we believe that it has several advantages. Beside the avoidance of the recursive computation of sub-frameworks, with the arguments in $\Delta_{F,S}$ one identifies for the whole framework the "defeated" arguments. Finally in the instance $[[F - \Delta_{F,S}]]$ one has at one glance the surviving arguments and attacks. The individual parts are easy to compute and intuitive. This characterization will facilitate further investigation steps such as an analysis of computational complexity (see Chapter 5), notions of equivalence (see Chapter 6) and of course the implementation in terms of ASP-encodings which was the initial motivation for the alternative characterization (see Chapter 7). In the next chapter we introduce *stage2*, a new semantics which combines the concepts of *cf2* and stage semantics to overcome some shortcomings of both of them as already mentioned in Section 2.2. In the course of this we will also exploit the alternative characterization to present the new *stage2* semantics.

CHAPTER 4

Incorporating Stage Semantics in the SCC-recursive Schema

In Section 2.2 we pointed out the special advantages of the two naive-based semantics stage and $cf2$. For instance the appearance of odd-length cycles and in particular self-attacking arguments as a special case of them, have a strong and sometimes undesired influence on the computation of solutions. None of the admissible-based semantics is able to select arguments of such a cycle as accepted, and moreover, they reject arguments just because they are attacked by a self-attacking argument. The reason for this behavior is that in an odd-length cycle, arguments defend their own attacker. As naive-based semantics do not rely on the notion of defense, one can accept both, arguments in an odd-length cycle, as well as arguments attacked by such arguments.

However, $cf2$ semantics treats odd-length cycles in a more sensitive way, the evaluation of odd-cycle-free (coherent) AFs e.g. if even-length cycles occur, is now questionable (see Section 2.2 and [64, 69]). On the other side, stage semantics [96] can also handle odd-length cycles and does not change the behavior of odd-cycle-free AFs. The disadvantages of stage semantics are that very basic properties are not satisfied, for example the skeptical acceptance of unattacked arguments, i.e. the weak reinstatement property [8] is violated (see Section 2.3).

While naive-based semantics seem to be the right candidates when the above described behavior of admissible-based semantics is unwanted, there are several shortcomings with existing approaches, as mentioned above. To overcome those problems we propose a new semantics combining concepts from $cf2$ and stage semantics, which we name $stage2$. This chapter is organized as follows.

- In Section 4.1 we combine the concepts of stage and $cf2$ semantics, where we use the SCC-recursive schema of $cf2$ semantics and instantiate the base case with stage semantics. In this way, we obtain the novel $stage2$ semantics. Furthermore, we prove that for $stage2$ one can also give an alternative characterization similar to the one for $cf2$.

- In Section 4.2 we point out the basic properties of the novel semantics and show that it overcomes most of the above mentioned problems. In particular, we compare $stage2$ with other semantics.

- In Section 4.3 we evaluate $stage2$ semantics with the criteria proposed by Baroni and Giacomin in [8].

- Finally in Section 4.4 we close the chapter with a short discussion on the novel $stage2$ semantics.

Parts of this chapter have been published in [44].

4.1 Combining Stage and $cf2$ Semantics

In Section 2.2, we observed that stage semantics has a more intuitive behavior on single SCCs than $cf2$ semantics. Whereas $cf2$ satisfies most of the general evaluation criteria.

Our suggestion is to combine the two semantics, where we use the SCC-recursive schema of the $cf2$ semantics and instantiate the base case with stage semantics. To retain the naming introduced by Baroni et al. in [12], we denote the obtained semantics as $stage2$.

Definition 24. *Let $F = (A, R)$ be an AF and $S \subseteq A$. Then, S is a $stage2$ extension of F, i.e. $S \in stage2(F)$, iff*

- *$S \in stage(F)$, in case $|SCCs(F)| = 1$;*

- *otherwise, $\forall C \in SCCs(F)$, $(S \cap C) \in stage2(F|_C - D_F(S))$.*

The only difference in the definition of $stage2$ compared to the one of $cf2$ (Definition 13) is that in the base case, where the AF consists of one SCC, the set S needs to be a stage extension. Whereas in the base case of $cf2$, S needs to be a naive extension of F. The remaining parts work equally to $cf2$, in particular $D_F(S)$ and the recursive computation of sub-frameworks is performed in the same way.

Let's consider the examples of Section 2.2, where both $cf2$ and stage produced questionable results. First we have a look at the AF from Example 10 illustrated in Figure 2.12 on page 20. F consists of one SCC, so S is a $stage2$ extension of F if S is a stage extension of F. Thus $stage2(F) = pref(F) = stable(F) = \{\{a, c, e\}, \{b, d, f\}\}$, whereas $cf2$ additionally accepts the naive sets $\{a, d\}, \{b, e\}$ and $\{c, f\}$. Remember in the case of Example 10, F has an even-length cycle.

Next we look at Example 7 depicted in Figure 2.7 on page 18. The AF F consists of two SCCs, $C_1 = \{a\}$ and $C_2 = \{b, c\}$. In this example $\{b\}$ is a stage extension although, b is attacked by the unattacked argument a. For $stage2$ we obtain the same result as for $cf2$, namely $\{a\}$ as the single extension. In this case, the computation of $stage2$ is exactly the same as for $cf2$, described in detail in Example 7.

These two examples showed that $stage2$ semantics is able to "repair" the undesired behavior of both, $cf2$ and stage semantics, but what happens with those AFs where we had nothing to bother, like the one from Example 5 (Figure 2.4 on page 15). In Example 6 on page 16 we already discussed the results for $cf2$ and stage semantics, where on this example they coincided. In this case $stage2$ semantics also results in the same extensions as the other naive-based semantics.

Alternative Characterization of $stage2$ Semantics

According to the alternative characterization of $cf2$ semantics, as introduced in Chapter 3, one can also formulate $stage2$ semantics in the same way.

Theorem 2. *For any AF F,*

$$stage2(F) = \{S \mid S \in naive(F) \cap stage([[F - \Delta_{F,S}]])\}.$$

The proof of Theorem 2 is similar to the one of Theorem 1, where another time we will make use of the set of recursively component defeated arguments $\mathcal{RD}_F(S)$ (Definition 21 on page 27). Lemma 7 gives the first alternative characterization of $stage2$.

Lemma 7. *Let $F = (A, R)$ be an AF and $S \subseteq A$. Then,*

$$S \in stage2(F) \textit{ iff } S \in stage([[F - \mathcal{RD}_F(S)]]).$$

Proof. We show the claim by induction over $\ell_F(S)$.
Induction base. For $\ell_F(S) = 1$, we have $|SCCs(F)| = 1$. By definition $\mathcal{RD}_F(S) = \emptyset$ and we have $[[F - \mathcal{RD}_F(S)]] = [[F]] = F$. Thus, the assertion states that $S \in stage2(F)$ iff $S \in stage(F)$ which matches the original definition for the $stage2$ semantics in case the AF has a single strongly connected component.
Induction step. Let $\ell_F(S) = n$ and assume the assertion holds for all AFs F' and sets S' with $\ell_{F'}(S') < n$. In particular, we have by definition that, for each $C \in SCCs(F)$, $\ell_{F|_C - D_F(S)}(S \cap C) < n$. By the induction hypothesis and Equations (3.2)-(3.5) (in the proof of Lemma 3 on page 28) we thus obtain that, for each $C \in SCCs(F)$ the following holds:

$$(S \cap C) \in stage2(F|_C - D_F(S)) \text{ iff } (S \cap C) \in stage\big([[F|_C - \mathcal{RD}_F(S)]]\big). \tag{4.1}$$

We now prove the assertion. Let $S \in stage2(F)$. By definition, for each $C \in SCCs(F)$, $(S \cap C) \in stage2(F|_C - D_F(S))$. Using (4.1), we get that for each $C \in SCCs(F)$, $(S \cap C) \in stage([[F|_C - \mathcal{RD}_F(S)]])$. By the definition of components and the semantics of stage, the following relation thus follows:

$$\bigcup_{C \in SCCs(F)} (S \cap C) \in stage\Big(\bigcup_{C \in SCCs(F)} [[F|_C - \mathcal{RD}_F(S)]]\Big).$$

Since $S = \bigcup_{C \in SCCs(F)} (S \cap C)$ and due to Lemma 2, $\bigcup_{C \in SCCs(F)} [[F|_C - \mathcal{RD}_F(S)]] = [[F - \mathcal{RD}_F(S)]]$, we arrive at $S \in stage([[F - \mathcal{RD}_F(S)]])$ as desired. The other direction is by essentially the same arguments. □

Proof of Theorem2. The result holds by the following observations. By Lemma 7, $S \in stage2(F)$ iff $S \in stage([[F - \mathcal{RD}_F(S)]])$. Moreover, due to Lemma 6, for any $S \in cf(F)$, $\Delta_{F,S} = \mathcal{RD}_F(S)$. Finally, $S \in stage2(F)$ implies $S \in naive(F)$. □

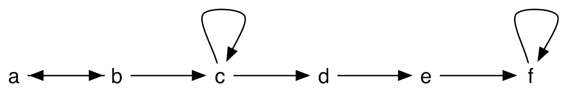

Figure 4.1: Framework F from Example 17.

4.2 Comparison of $stage2$ with other Semantics

The novel $stage2$ semantics is clearly a naive-based semantics due to the way it is defined. In this section we compare $stage2$ with other naive-based semantics w.r.t. the $\subseteq$-relations between the sets of extensions. Furthermore, we consider coherent AFs, as stage semantics also coincides with stable and preferred on these frameworks but $cf2$ does not.

We start with stage and $stage2$ semantics which are in general incomparable w.r.t. set inclusion. For instance, consider the following example.

Example 17. *Let $F = (A, R)$ as illustrated in Figure 4.1. Then, the naive sets of F are $\{a, d\}$, $\{a, e\}$, $\{b, d\}$ and $\{b, e\}$. We consider first stage semantics, therefore we compute the range of each naive set.*

- $\{b, d\}^+_R = \{a, b, c, d, e\}$,
- $\{b, e\}^+_R = \{a, b, d, e, f\}$,
- $\{a, d\}^+_R = \{a, b, d, e\} \subset \{b, e\}^+_R$,
- $\{a, e\}^+_R = \{a, b, e, f\} \subset \{b, e\}^+_R$.

Thus, $stage(F) = \{\{b, d\}, \{b, e\}\}$.

The $stage2$ extensions are $\{a, d\}$ and $\{b, d\}$ which are computed as follows.

- *For $S_1 = \{a, d\}$, $\Delta_{F,S_1} = \{e\}$ and $S_1 \in stage([[F - \Delta_{F,S_1}]])$. Thus, $S_1 \in stage2(F)$.*
- *For $S_2 = \{b, d\}$, $\Delta_{F,S_2} = \{c, e\}$ and $S_2 \in stage([[F - \Delta_{F,S_2}]])$. Thus, $S_2 \in stage2(F)$.*
- *For $S_3 = \{a, e\}$, $\Delta_{F,S_3} = \{f\}$ but $S_3 \notin stage([[F - \Delta_{F,S_3}]])$ because $S^+_{3_{R'}} = \{a, b, e\}$ and there is the set $T \in naive(F')$ with $T = \{a, d, e\}$ and $T^+_{R'} = \{a, b, d, e\} \supset S^+_{3_{R'}}$ where $F' = [[F - \Delta_{F,S_3}]]$. Hence, $S_3 \notin stage2(F)$.*
- *For $S_4 = \{b, e\}$, $\Delta_{F,S_4} = \{c, f\}$ but $S_4 \notin stage([[F - \Delta_{F,S_4}]])$ because $S^+_{4_{R''}} = \{a, b, e\}$ and there is the set $T \in naive(F'')$ with $T = \{a, d, e\}$ and $T^+_{R''} = \{a, b, d, e\} \supset S^+_{4_{R''}}$ where $F'' = [[F - \Delta_{F,S_4}]]$. Hence, $S_4 \notin stage2(F)$.*

$\diamond$

Now, we consider the relation between $cf2$ and $stage2$ semantics. By Example 10 we know that there are AFs with $cf2(F) \not\subseteq stage2(F)$.

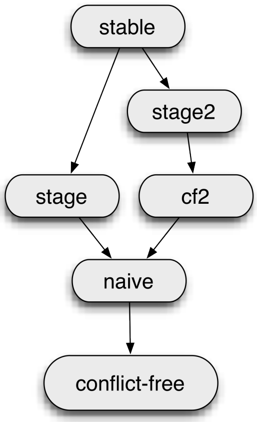

Figure 4.2: Relations between naive-based semantics

Proposition 1. *For any AF $F = (A, R)$, $stage2(F) \subseteq cf2(F)$.*

Proof. Consider a set $S \in stage2(F)$. By Theorem 2, $S \in naive(F) \cap stage([[F - \Delta_{F,S}]])$. Now using that for every AF G, $stage(G) \subseteq naive(G)$ we obtain $S \in naive(F) \cap naive([[F - \Delta_{F,S}]])$. By Theorem 1, $S \in cf2(F)$. □

Next, we study the relations between stable and $stage2$ semantics.

Proposition 2. *For any AF $F = (A, R)$, $stable(F) \subseteq stage2(F)$.*

Proof. Consider $E \in stable(F)$, from Lemma 1 we know that $E \in naive(F)$ and for each $a \in A \setminus E$ there exists $b \in E$ such that $(b, a) \in R$. Hence, $a \in E^{+}_{R_F}$. It remains to show that $E \in stage([[F - \Delta_{F,E}]])$. We show the stronger statement $E \in stable([[F - \Delta_{F,E}]])$.
To this end, let $F' = F - \Delta_{F,E}$ and $F'' = [[F - \Delta_{F,E}]]$, we have either $a \in \Delta_{F,E}$ or $a \in A_{F'}$. For $a \in A_{F'} = A_{F''}$, we need to show that $a \in E^{+}_{R_{F''}}$. If $a \in E$ clearly $a \in E^{+}_{R_{F''}}$, hence we consider $a \in A_{F'} \setminus E$. As E is stable there exists $b \in E$ such that $(b, a) \in R_{F'}$. Now as $a \notin \Delta_{F,E}$, by Definition 23 we know that $a \Rightarrow_{F}^{A \setminus \Delta_{F,E}} b$. In other words a, b are in the same SCC of F' and thus $(b, a) \in R_{F''}$. Hence, for every $a \in A_{F''} \setminus E$ there is an argument $b \in E$ such that $(b, a) \in R_{F''}$, hence $E \in stable(F'')$. As for any AF G $stable(G) \subseteq stage(G)$, it follows that $E \in stage(F'')$. Thus, by Theorem 2, $E \in stage2(F)$. □

Figure 4.2 gives an overview of the relations between naive-based semantics. An arrow from semantics σ to semantics τ encodes that each σ extension is also a τ extension. Furthermore, if there is no directed path from σ to τ, then one can construct AFs with a σ extension that is not a τ extension.

Now we turn to frameworks which have special properties. We start with AFs which possess at least one stable extension then, stage coincides with stable semantics. Obviously, this does not hold for $stage2$ semantics.

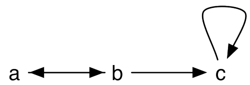

Figure 4.3: AF F from Example 18.

Example 18. *Consider the AF* $F = (\{a,b,c\}, \{(a,b), (b,a), (b,c), (c,c)\})$ *depicted in Figure 4.3. We obtain* $stage2(F) = \{\{a\},\{b\}\}$ *and* $stable(F) = \{\{b\}\}$. $\diamond$

However, these semantics comply with each other in *coherent* AFs, i.e. AFs where stable and preferred semantics coincide.

Proposition 3. *For any coherent AF F,* $stable(F) = stage(F) = stage2(F)$.

Proof. By Proposition 2, $stable(F) \subseteq stage2(F)$ and thus it only remains to show that also $stable(F) \supseteq stage2(F)$ holds for each coherent AF F.

Let us first consider the case where F consists of a single SCC. Then, $stage2$ semantics coincides with stage semantics and as F is coherent also with stable semantics.

Now, let this be our induction base, and let us assume the claim holds for AFs of size $< n$. Let us consider an AF F of size n with $(C_i)_{1\leq i\leq m}$ being the SCCs of F, such that there is no attack from C_i to C_j for $j < i$. If $m = 1$ we are in the base-case, hence let us assume that $m \geq 2$. Consider $S \in stage2(F)$ and $S_1 = S \cap \bigcup_{1\leq i<m} C_i$, $S_2 = S \cap C_m$. By definition of $stage2$ we know that $S_1 \in stage2(F - C_m)$ and $S_2 \in stage2(F|_{C_m} - S_1^+)$. Note, $S_1 \cap S_2 = \emptyset$. By assumption, F is coherent and it is easy to see that also $F - C_m$ is coherent. Hence, by the induction hypothesis, $stable(F - C_m) = pref(F - C_m) = stage2(F - C_m)$.

Next, we show that also $F|_{C_m} - S_1^+$ is coherent. By definition, $stable(F) \subseteq pref(F)$. Now, consider an extension $E_2 \in pref(F|_{C_m} - S_1^+)$. By the directionality of $pref$ and the fact that $S_1 \in stable(F - C_m)$, we obtain $(S_1 \cup E_2) \in pref(F)$. Now, as F is coherent also $(S_1 \cup E_2) \in stable(F)$ and thus, $E_2 \in stable(F|_{C_m} - S_1^+)$. Hence, $F|_{C_m} - S_1^+$ is coherent and again we can use the induction hypothesis.

Finally, we obtain $S_1 \in stable(F - C_m)$ and $S_2 \in stable(F|_{C_m} - S_1^+)$, combining these results we get $S \in stable(F)$. $\square$

Notice, the last proposition implies that on coherent AFs $stage2$ semantics coincides with preferred, stage and semi-stable [31] semantics, because on coherent AFs all these semantics coincide with stable semantics.

4.3 Evaluation Criteria w.r.t. $stage2$ Semantics

To continue the systematic analysis for $stage2$ semantics we consider in this section the extension and skepticism evaluation criteria as already discussed in Section 2.1. Remember, concerning the extension evaluation criteria $cf2$ semantics satisfies I-maximality, weak- and $\mathcal{CF}$-reinstatement as well as directionality. Whereas, for stage semantics we only knew that

I-maximality is satisfied but reinstatement and directionality are not fulfilled. Regarding the skepticism evaluation criteria no results for naive and stage semantics were known. Therefor, we not only consider $stage2$ semantics, but we also give the missing results for naive and stage semantics.

We start with some general extension evaluation properties of naive-based semantics.

Proposition 4. *I-maximality and $\mathcal{CF}$-reinstatement are satisfied by each semantics σ with $\sigma(F) \subseteq naive(F)$.*

Proof. Clearly naive semantics satisfies both I-maximality and $\mathcal{CF}$-reinstatement. A set E which is $\subseteq$-maximal in $naive(F)$ is also maximal in each subset of $naive(F)$ and thus, σ satisfies I-maximality. $\mathcal{CF}$-reinstatement is a property defined on single extensions, and as each σ-extension is also a naive extension, $\mathcal{CF}$-reinstatement is satisfied. $\square$

Among the naive-based semantics, only stable semantics satisfies the reinstatement property, which is due to the fact that it is also an admissible-based semantics.

Proposition 5. *The reinstatement property is not satisfied by semantics which can select non-empty conflict-free subsets out of odd-length cycles.*

Proof. Consider an odd-length cycle $F = (\{a_1, \ldots, a_n\}, \{(a_i, a_{i+1 \bmod n}) \mid 1 \leq i \leq n\})$ with n being an odd integer. We claim that no $E \in cf(F)$ and $E \neq \emptyset$ satisfies the reinstatement property. Now, towards a contradiction let us assume there exists a nonempty $E \in cf(F)$ satisfying the reinstatement property. W.l.o.g. assume that $a_1 \in E$. Then a_3 is defended and by assumption $a_3 \in E$. But then also a_5 is defended, and by induction it follows that $a_i \in E$ if i is odd. Hence also $a_n \in E$, but $\{a_1, a_n\} \subseteq E$ contradicts that E is conflict-free in F. $\square$

It follows that naive, stage, *cf2* and $stage2$ semantics do not satisfy the reinstatement criterion. An example for the *cf2* semantics and the unsatisfied reinstatement criterion is a simple AF consisting of an odd-length cycle with length three.

For instance, consider the AF $F = (\{a, b, c\}, \{(a, b), (b, c), (c, a)\})$. By the conflict-freeness of extensions we can just select a single argument but this argument defends an other argument, for instance the set $\{a\}$ defends argument c. Hence, when considering naive-based semantics we are usually interested in weaker forms of reinstatement, namely the weak- or $\mathcal{CF}$-reinstatement.

Proposition 6. *The weak reinstatement and directionality criterion are not satisfied by naive and stage semantics.*

Proof. Consider the AF F from Example 1. We obtain $naive(F) = stage(F) = \{\{a\}, \{b\}\}$ and the grounded extension $G = \{a\}$. Then, the weak reinstatement criterion is not satisfied because $G \not\subseteq \{b\}$. Now let us consider directionality and the sub-framework $F|_{\{a\}}$. Then $stage(F|_{\{a\}}) = \{\{a\}\}$ but $\{(\{a\} \cap S) \mid S \in stage(F)\} = \{\emptyset, \{a\}\}$, contradicting the directionality criterion. $\square$

Proposition 7. *The weak reinstatement criterion is satisfied by $stage2$ semantics.*

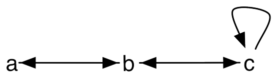

Figure 4.4: Framework F.

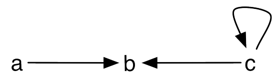

Figure 4.5: Framework G.

Proof. Let $F = (A, R)$ and $E \in grd(F)$. Due to [12], for any AF F and any $S \in cf2(F)$, $E \subseteq S$. From Proposition 1 we know that for any AF G, $stage2(G) \subseteq cf2(G)$. It follows that for any extension $S \in stage2(F)$, $S \in cf2(F)$ and $E \subseteq S$. $\square$

Next we consider the skepticism related criteria for the naive-based semantics, where we complete Table 2.1 for stage, naive and $stage2$ semantics.

Proposition 8. *Stage and $stage2$ semantics are not $\preceq^E_\cap$-skepticism adequate.*

Proof. Consider the AFs F and G of Figure 4.4 and 4.5.
We start with the proof for stage semantics. According to Definition 18, we need to show that for any two AFs F and G, such that $F \preceq^A G$, $stage(F) \preceq^E_\cap stage(G)$ holds. $F \preceq^A G$ clearly holds as $A(F) = A(G)$, $conf(F) = conf(G)$ and $R(G) \subseteq R(F)$. Due to Definition 16, $stage(F) \preceq^E_\cap stage(G)$ iff $\bigcap_{S_1 \in stage(F)} S_1 \subseteq \bigcap_{S_2 \in stage(G)} S_2$. But $stage(F) = \{\{b\}\}$ and $stage(G) = \{\{a\}\}$. Hence, as $\{b\} \not\subseteq \{a\}$ the condition for $\preceq^E_\cap$-skepticism adequacy is not satisfied. The $stage2$ extensions of these two AFs are exactly the same as for stage semantics, so the argumentation of the proof for $stage2$ is the same. $\square$

As the weakest form of skepticism adequacy is not satisfied by stage and $stage2$ semantics, therefor also the stronger version, $\preceq^E_W$-skepticism adequacy is not satisfied.

For naive sets we have the following observation.

Proposition 9. *For any AFs F and G, $conf(F) = conf(G)$ iff $naive(F) = naive(G)$.*

Proof. Since for any AFs F and G, $conf(F) = conf(G)$ obviously implies $cf(F) = cf(G)$, and as two AFs with the same conflict-free sets also coincide with the naive extensions, it follows that $conf(F) = conf(G)$ implies $naive(F) = naive(G)$. The same argument holds for the other direction. $\square$

From Proposition 9 it follows that naive sets satisfy all skepticism-adequacy criteria.
We summarize the evaluation criteria w.r.t. naive-based semantics in Table 4.1.

4.4 Discussion of $stage2$ Semantics

In this chapter we proposed the new semantics $stage2$ which combines concepts of $cf2$ and stage to overcome their shortcomings. We provided a broad discussion of $stage2$, its properties and relations to other semantics. First, beside the definition via the SCC-recursive schema we provided an alternative characterization which is similar to that of $cf2$ semantics and thus allows

	naive	*stable*	*stage*	*cf2*	*stage2*
I-max.	Yes	Yes	Yes	Yes	Yes
Reinst.	No	Yes	No	No	No
Weak reinst.	No	Yes	No	Yes	Yes
$\mathcal{CF}$-reinst.	Yes	Yes	Yes	Yes	Yes
Direct.	No	No	No	Yes	Yes
$\preceq^E_\cap$-sk. ad.	Yes	Yes	No	Yes	No
$\preceq^E_W$-sk. ad.	Yes	Yes	No	Yes	No

Table 4.1: Evaluation Criteria w.r.t. Naive-based Semantics.

to extend several results for $cf2$ also to $stage2$. Further, we showed that $stage2$ fixes the shortcomings of stage semantics w.r.t. the extension evaluation criteria proposed by [8]. We related $stage2$ semantics to the existing semantics showing that $stable(F) \subseteq stage2(F) \subseteq cf2(F)$. Moreover, we observed that on coherent AFs $stage2$ semantics coincides with stable and preferred semantics.

CHAPTER 5

Complexity Analysis

Abstract argumentation frameworks are formalized in a simple way but their evaluation require involved concepts. As we saw in the previous chapters they can be represented as directed graphs. The challenging part lies in the semantic evaluation. Depending on which semantics is chosen, the related reasoning problems can be very hard. For example, deciding whether an argument is acceptable in at least one preferred extensions is known to be NP-complete.

> *Computational complexity theory deals with classifying computational problems with respect to the resources needed for their solution, e.g., the time required by the fastest program that will solve the problem. (Dunne and Wooldridge [43])*

The study of computational complexity is very important for the analysis of argumentation semantics, as it gives upper and lower bounds for specific reasoning problems. This then provides the basis for suitable algorithms to solve the problems.

Computational complexity has been studied for many argumentation semantics. An overview can be found in [43] as well as in the doctoral thesis of Wolfgang Dvořák [47]. However, regarding $cf2$ semantics the only mentionable reference in this context is the article of Nieves et al. [82], where the authors state that the decision problem of verifying if a set is a $cf2$ extension (Ver_{cf2}) is in P. To complete the analysis of computational complexity also for $cf2$ and $stage2$ semantics we will study the standard reasoning problems (which will be introduced in the following) for these two semantics. The alternative characterizations, as formulated in Chapter 3 and Chapter 4, will facilitate the analysis. As all those complexity results are worst-case complexity we will also investigate possible tractable fragments, i.e. instances of the argumentation frameworks which are easier to solve. We make use of these results in Chapter 7 where we introduce a labeling-based algorithm for $cf2$ and $stage2$ semantics.

This chapter is organized as follows.

- First, in Section 5.1 we recapitulate some basic concepts of computational complexity and introduce the needed complexity classes.

- Then, in Section 5.2 we give the main reasoning problems for abstract argumentation, summarize known complexity results and investigate the complexity of the introduced reasoning problems for $cf2$ and $stage2$ semantics.
- In Section 5.3 we consider tractable fragments for $cf2$ and $stage2$ semantics.
- Finally, in Section 5.4 we discuss the achieved results and future directions.

Parts of this chapter have been published in [44, 45, 69].

5.1 Background of Computational Complexity

This section is based on the summary given by by Dunne and Wooldridge in [43] and the standard work of Papadimitriou [87]. For a more detailed description we refer to the respective references.

Basic Concepts

When we speak about computational problems we normally refer to *decision problems* which can be divided into *instances* and the *question* asked of these instances. A well studied decision problem in this context is *3-CNF Satisfiability (3-SAT).*

Definition 25. *A propositional formula φ is in* 3-Conjunctive Normal Form (3-CNF)*, if $\varphi = \bigwedge_{j=1}^{m} C_j$ is given over atoms $Z = \{z_1, \ldots, z_n\}$ with $C_j = l_{j1} \vee l_{j2} \vee l_{j3}$, where $(1 \leq j \leq m)$ and l_{jk} is a literal from Z.*
3-SAT *is the satisfiability problem of a 3-CNF formula φ, i.e. the question: is there a set $M \subseteq Z$ satisfying φ, or is there a truth assignment to the variables in Z such that the formula φ evaluates to true?*

Example 19. *Let $\varphi = (z_1 \vee z_2 \vee z_3) \wedge (\neg z_2 \vee \neg z_3 \vee \neg z_4) \wedge (\neg z_1 \vee z_2 \vee z_4)$. Then, the set $M = \{z_1, z_2\}$ is a model of φ i.e., the assignment $z_1 = true, z_2 = true, z_3 = false$ and $z_4 = false$ is a* YES*-instance of φ. Thus, φ is 3-SAT.* $\diamond$

Before we introduce the different complexity classes we need to define the concept of a Turing machine, which is needed for the classification of the decision problems. Such a Turing machine is designed to express any algorithm and simulate any programming language.

Definition 26. *A (deterministic) k-string Turing machine (TM) is a tuple $M = (K, \Sigma, \delta, s)$, where*

- *K is a finite set of states;*
- *$s \in K$ is the initial state;*
- *Σ is the alphabet of M - a finite set of symbols;*
- *$\delta : K \times \Sigma^k \mapsto K \cup \{$“yes”, “no”$\} \times \Sigma^k \times \{\leftarrow, \rightarrow, -\}^k$ is a transition function.*

Initially the state is s and the cursor points to the first symbol on the tape. Then, according to δ, the machine changes its state, prints a symbol and moves the cursor. This is repeated till one of the halting states "yes" or "no" is reached, where the former state *accepts* the input and the latter *rejects* the input.

Next, we generalize the concept of a Turing machine to non-deterministic and oracle Turing machines.

Definition 27. *A non-deterministic k-string Turing machine is a quadruple $N = (K, \Sigma, \Delta, s)$, with K, Σ and s as for an ordinary Turing machine, where the transition relation Δ gives a choice between several next actions, i.e.*

$$\Delta \subseteq (K \times \Sigma^k) \times [(K \cup \{\text{"yes", "no"}\}) \times \Sigma^k \times \{\leftarrow, \rightarrow, -\}^k].$$

In contrast to a deterministic Turing machine, where in each configuration there is only one computation step, a non-deterministic Turing machine has several possible computation steps, and it accepts the input if at least one of the possible computations accepts it, and it rejects the input if all possible computations reject it.

Finally we define a $\mathcal{C}$-oracle machine which is a Turing machine that can access an oracle that decides a (sub)-problem $\mathcal{C}$ in one step.

Definition 28. *For a language $\mathcal{L}$, an $\mathcal{L}$-oracle Turing machine is a (non-deterministic) k-string Turing machine with an designated query string and three special states $q_?$, q_{yes} and q_{no}. The state $q_?$ is excluded from the function (resp. relation) δ. The transition step for a configuration with state $q_?$ is handled by the $\mathcal{L}$-oracle. The state changes to q_{yes} if the current string on the query string is in $\mathcal{L}$ and to q_{no} otherwise. The strings as well as the heads positions are not changed in this step.*

Complexity Classes

In computational complexity theory, problems are divided into classes requiring the same resources. In our case we are mainly interested in the time required to solve a problem.

Given a language L, deciding whether $x \in L$ can be solved by constructing a program M such that for a constant value k,

- if $x \in L$, then M returns *"accept"*, else M returns *"reject"*;
- M terminates after at most $|x|^k$ steps.

Then, the program M provides an algorithm for L with run-time n^k which leads to the complexity class of *polynomial time decidable* languages P. In the following we formulate this in terms of Turing machines.

The class P is the class of problems which can be decided by a deterministic Turing machine in polynomial time. Problems in the complexity class P are generally regarded to be computationally easy or *tractable*. Next we consider the class NP (non-deterministic polynomial time) which is the class of problems decidable by a non-deterministic Turing machine in polynomial time. Problems in NP are called *intractable*, for example the 3-SAT problem as discussed above is a typical problem falling into the class NP.

Each problem in NP *has a remarkable property: Any "yes" instance x of the problem has at least one succinct certificate (or polynomial witness) y of its being a "yes" instance. Naturally "no" instances possess no such certificates. We may not know how to discover this certificate in polynomial time, but we are sure it exists if the instance is a "yes" instance. (Papadimitriou [87])*

These problems are often solved by first guessing all certificates and then checking each of them to be a "yes" instance. Then, the non-deterministic part is the guessing and the checking can be done in polynomial time. In the worst case one needs to check each guess to answer the decision problem. If the instances are very big the procedure may not terminate in reasonable time. Thus, the problem remains unsolved. This Guess&Check methodology is normally used in answer-set programming (ASP) which we will discuss in Chapter 7.

The class coNP is the class of problems $\mathcal{X}$ where the complement $\bar{\mathcal{X}}$ can be decided by a non-deterministic Turing machine in polynomial time. The 3-UNSAT problem, i.e., if a 3-CNF formula is unsatisfiable, is known to be in coNP.

The class EXPTIME (exponential time) is the class of problems that can be solved by a deterministic Turing machine in exponential time. The class PSPACE (deterministic polynomial space) is the class of problems that can be decided by a deterministic Turing machine in polynomial space and exponential time.

$\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$ is the class of problems which can be decided by a non-deterministic polynomial time algorithm that has access to an NP-oracle. $\Pi_2^{\mathrm{P}} = \mathrm{coNP}^{\mathrm{NP}}$ is the class of problems where the complement can be decided by a non-deterministic polynomial time algorithm that has access to an NP-oracle.

To classify problems we need the term *reduction*. We say a problem A is at least as hard as problem B if B reduces to A. This means, there is a transformation R which produces for every input x of B, an equivalent input $R(x)$ of A. So, to solve B on input x we have to compute $R(x)$ and solve A on it. As we want to compare time classes, the reductions need to be polynomial-time algorithms. Then, B is *hard* for a complexity class $\mathcal{C}$ if for any $A \in \mathcal{C}$, A is polynomial-time reducible to B. A hardness result for a problem provides an upper bound. Furthermore, B is *$\mathcal{C}$-complete* if B is $\mathcal{C}$-hard and $B \in \mathcal{C}$. Then, completeness of a problem for a complexity class stands for a lower bound, i.e. that the problem can not be solved with an algorithm situated in a lower complexity class.

The relation between the introduced complexity classes is as follows:

$$\mathrm{P} \subseteq \begin{matrix}\mathrm{NP}\\ \mathrm{coNP}\end{matrix} \subseteq \begin{matrix}\Sigma_2^{\mathrm{P}}\\ \Pi_2^{\mathrm{P}}\end{matrix} \subseteq \text{PSPACE} \subseteq \text{EXPTIME}.$$

It is known that P is a proper subset of EXPTIME. On the other side, there are still many open questions. The most prominent of them is $\mathrm{P} = \mathrm{NP}$?

5.2 Complexity of Abstract Argumentation

In this section we study computational complexity of abstract argumentation, to be more precise, we first formulate the standard reasoning problems for argumentation semantics and summarize

	Ver_σ	$Cred_\sigma$	$Skept_\sigma$	NE_σ
naive	in P	in P	in P	in P
grd	P-c	P-c	P-c	in P
stable	in P	NP-c	coNP-c	NP-c
adm	in P	NP-c	trivial	NP-c
compl	in P	NP-c	P-c	NP-c
*grd**	P-c	NP-c	coNP-c	in P
pref	coNP-c	NP-c	Π_2^{P}-c	NP-c
stage	coNP-c	Σ_2^{P}-c	Π_2^{P}-c	in P
semis	coNP-c	Σ_2^{P}-c	Π_2^{P}-c	NP-c

Table 5.1: Complexity of decision problems ($\mathcal{C}$-c denotes completeness for class $\mathcal{C}$).

known results for the introduced semantics. Then, we investigate the complexity of *cf2* and *stage2* semantics.

Decision Problems in Abstract Argumentation

We consider the following decision problems for given $F = (A, R)$, a semantics σ, $a \in A$ and $S \subseteq A$:

- Verification Ver_σ: is $S \in \sigma(F)$?
- Credulous acceptance $Cred_\sigma$: is a contained in at least one σ extension of F?
- Skeptical acceptance $Skept_\sigma$: is a contained in every σ extension of F?
- Non-emptiness NE_σ: is there any $S \in \sigma(F)$ for which $S \neq \emptyset$?

In Table 5.1 known complexity results are summarized[5]. For a detailed analysis of them we refer to [47] as well as to [34, 36, 39, 41, 42, 46]. All these results are understood as worst-case complexity.

Complexity of *cf2* Semantics

So far, the complexity of *cf2* semantics has not been studied, except for a note in [82], where the authors state that the decision problem Ver_{cf2} is in P. In the following we proof this statement with the help of our alternative characterization.

Theorem 3. *Ver_{cf2} is in* P.

[5] We omit the ideal and eager semantics because for them different reasoning problems are related, which are out of the scope of this work. For the interested reader we refer to [47].

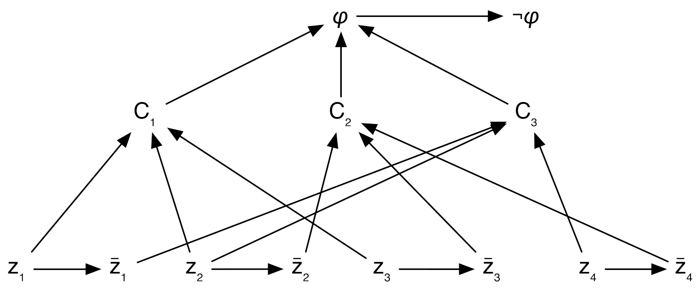

Figure 5.1: AF F_φ for the example 3-CNF φ.

Proof. For any AF $F = (A, R)$ and a set $S \subseteq A$, to check if $S \in cf2(F)$ can be computed in polynomial time. We show that all steps in Theorem 1 (on page 32) are in P. Verifying if $S \in naive(F)$ can be done in polynomial time [34]. Given $\Delta_{F,S}$, computing the instance $[[F - \Delta_{F,S}]]$ can be done efficiently; this follows from known results about graph reachability and efficient algorithms for computing SCCs [94]. It remains to show that the operator $\Delta_{F,S}(D)$ reaches its fixed-point after a polynomial number of iterations. The operator is clearly monotonic, and it is easy to see that in every iteration less or equal connections between the arguments do exist. Hence, the computation terminates when no argument a is attacked by any $b \in S$, and $a \not\Rightarrow_F^{A \setminus D} b$. □

For the hardness proofs of $Cred_{cf2}$ and $Skept_{cf2}$ we use the standard reduction from propositional formulas in CNF to AFs as in [36, 39].

Definition 29. *Given a 3-CNF formula $\varphi = \bigwedge_{j=1}^{m} C_j$ over atoms Z with $C_j = l_{j1} \vee l_{j2} \vee l_{j3}$ ($1 \leq j \leq m$) the corresponding AF $F_\varphi = (A_\varphi, R_\varphi)$ is built as follows.*

$$\begin{aligned} A_\varphi &= Z \cup \bar{Z} \cup \{C_1, \ldots, C_m\} \cup \{\varphi\} \cup \{\neg\varphi\} \\ R_\varphi &= \{(z, \bar{z}), (\bar{z}, z) \mid z \in Z\} \cup \{(C_j, \varphi) \mid j \in \{1, \ldots, m\}\} \cup \{(\varphi, \neg\varphi)\} \cup \\ &\quad \{(z, C_j) \mid j \in \{1, \ldots, m\}, z \in \{l_{j1}, l_{j2}, l_{j3}\}\} \cup \\ &\quad \{(\bar{z}, C_j) \mid j \in \{1, \ldots, m\}, \neg z \in \{l_{j1}, l_{j2}, l_{j3}\}\} \end{aligned}$$

Figure 5.1 illustrates the AF F_φ of the formula

$$\varphi = (z_1 \vee z_2 \vee z_3) \wedge (\neg z_2 \vee \neg z_3 \vee \neg z_4) \wedge (\neg z_1 \vee z_2 \vee z_4).$$

Lemma 8. *For any cf2 extension E of the AF $F_\varphi = (A_\varphi, R_\varphi)$ and $z_i \in Z$ for $i \in \{1, \ldots, n\}$, either $z_i \in E$ or $\bar{z}_i \in E$.*

Proof. The AF F_φ has the following singleton SCCs $\{\varphi\}$, $\{\neg\varphi\}$, and $\{C_j\}$ ($1 \leq j \leq m$). The remaining SCCs are $S_i \in \{S_1, \ldots, S_n\}$, with $S_i = \{z_i, \bar{z}_i\}$. As all S_i are not attacked from outside their component they remain unchanged in $[[F_\varphi - \Delta_{F_\varphi,E}]]$ and $naive(F_\varphi|_{S_i}) = \{\{z_i\}, \{\bar{z}_i\}\}$. Hence, either $z_i \in E$ or $\bar{z}_i \in E$ (but never both). □

Theorem 4. *$Cred_{cf2}$ is* NP*-complete.*

Proof. For hardness, we show that any 3-CNF formula φ is satisfied iff the corresponding AF F_φ (as in Definition 29) has a *cf2* extension containing φ.
For the if direction, let φ be a 3-CNF formula over Z and $M \subseteq Z$ a model of φ. We show that

$$E = \{z_i \mid z_i \in M\} \cup \{\bar{z}_i \mid z_i \in Z \setminus M\} \cup \{\varphi\}$$

is a *cf2* extension of F_φ. We need to show

(i) E is a naive extension of F_φ and

(ii) $E \in naive([[F_\varphi - \Delta_{F_\varphi,E}]])$.

Ad (i), from Lemma 8 we know that for all $i \in \{1,\ldots,n\}$ either z_i or $\bar{z}_i$ is in E, so there are no conflicts between the arguments in Z and $\bar{Z}$. The argument φ is not attacked by any z_i at all. Hence, it is easy to see that $E \in naive(F_\varphi)$.
Ad (ii), let us first compute $\Delta_{F_\varphi,E}$, where

$$\Delta_{F_\varphi,E}(\emptyset) = \{x \in A_\varphi \mid \exists l \in E : l \neq x, (l,x) \in R_\varphi, x \not\Rightarrow^A_F l\}.$$

As M is a model of φ, all clauses in φ are satisfied, hence, for each C_j there is an l_i such that $(l_i, C_j) \in R_\varphi$, where $l_i \in \{z_i, \bar{z}_i\}$ for $j = \{1,\ldots,m\}$ and $i = \{1,\ldots,n\}$. Furthermore, $\varphi \in E$, $(\varphi, \neg\varphi) \in R_\varphi$ and $\neg\varphi \not\Rightarrow \varphi$. Therefore, we obtain $\Delta_{F_\varphi,E}(\emptyset) = \{C_1,\ldots,C_m,\neg\varphi\}$ which is also the lfp $\Delta_{F_\varphi,E}$. Finally, we compute the instance

$$[[F_\varphi - \Delta_{F_\varphi,E}]] = (A_\varphi \setminus \{C_1,\ldots,C_m,\neg\varphi\}, \{(z,\bar{z}),(\bar{z},z) \mid z \in Z\}).$$

It is easy to see that $E \in naive([[F_\varphi - \Delta_{F_\varphi,E}]])$ holds.
Only if: Let $E \in cf2(F_\varphi)$ such that $\varphi \in E$. We show that $M = \{z_i \mid z_i \in E\}$ is a model of φ. As $\varphi \in E$ we know it is not attacked by any $d \in \Delta_{F_\varphi,E}$. Assume there exists a $C_j \notin \Delta_{F_\varphi,E}$ with $(C_j, \varphi) \in R_\varphi$. We know $C_j \notin E$ because $E \in naive(F_\varphi)$, hence from Definition 23 we conclude there is no $x \in E$ such that $(x, C_j) \in R_\varphi$. In this case, the argument C_j is contained in $[[F_\varphi - \Delta_{F_\varphi,E}]]$, but this is a contradiction to $E \in naive([[F_\varphi - \Delta_{F_\varphi,E}]])$, because the set $E' = E \cup \{C_j\}$ is conflict-free in $[[F_\varphi - \Delta_{F_\varphi,E}]]$. It follows that for each C_j there exists a $l_i \in \{z_i, \bar{z}_i\}$ such that $(l_i, C_j) \in R_\varphi$, for $j = \{1,\ldots,m\}$. This means that for every clause C_j there exists a literal $l_i \in M$. Hence, M is a model of φ.
For membership one can construct an algorithm as follows. For any AF $F = (A, R)$ and $a \in A$, guess $S \subseteq A$ with $a \in S$ and check $S \in cf2(F)$. As $Ver_{cf2} \in$ P, this yields an NP algorithm. □

Theorem 5. *$Skept_{cf2}$ is* coNP*-complete.*

Proof. For hardness, we show that a given 3-CNF formula φ is unsatisfiable iff $\neg\varphi$ is contained in every *cf2* extension of F_φ, where F_φ is constructed following Definition 29.
For the if direction, let $E \in cf2(F_\varphi)$ such that $\neg\varphi \in E$. If $\neg\varphi \in E$ and as $(\varphi, \neg\varphi) \in R_\varphi$

we can conclude that $\varphi \in \Delta_{F_\varphi,E}$, hence there exists a $C_j \in E$ such that $(C_j, \varphi) \in R_\varphi$. From the proof of Theorem 4 we know, if φ is satisfiable then $C_j \notin E$ for each $C_j \in \{C_1, \dots, C_m\}$ hence, φ is unsatisfiable.

Only if: Let $E \in cf2(F_\varphi)$ such that $\neg\varphi \notin E$. We show that φ is satisfiable. The only reason for $\neg\varphi \notin E$ is $\neg\varphi \in \Delta_{F_\varphi,E}$. As φ is the only argument attacking $\neg\varphi$, we obtain $\varphi \in E$. In the proof of Theorem 4 we already showed that if $\varphi \in E$ then φ is satisfied.

Membership can be shown as follows via the complementary problem. Thus, for given a AF $F = (A, R)$ and $a \in A$ we guess a set S with $a \notin S$ and check $S \in cf2(F)$. As $Ver_{cf2} \in \mathrm{P}$, this yields an NP algorithm for the complementary problem of $Skept_{cf2}$. Thus, we obtain that $Skept_{cf2}$ is in coNP. □

Theorem 6. *NE_{cf2} is in* P

Proof. Recall, for every AF F it holds that each $cf2$ extension of F is a naive extension of F. Thus, in case we have an F which possesses only the empty set as its $cf2$ extension, we know, the empty set is also the only naive extension of F. However, this is only the case if all arguments of F are self-attacking. Thus, to decide whether there exists a non-empty $cf2$ extension of an AF $F = (A, R)$, it is sufficient to check if there exists any argument $a \in A$ such that $(a, a) \notin R$. This can be done in polynomial time. □

Complexity of $stage2$ Semantics

Theorem 7. *For stage2 semantics the following holds*

- Ver_{stage2} *is* coNP*-complete;*
- $Cred_{stage2}$ *is* Σ_2^{P}*-complete;*
- $Skept_{stage2}$ *is* Π_2^{P}*-complete;*
- NE_{stage2} *is in* P.

Proof. We first consider the membership part starting with Ver_{stage2}. Given an AF $F = (A, R)$ a set E of arguments, by Proposition 2 (on page 37) we have to check whether $E \in naive(F)$ (which can be done in P), and whether $E \in stage([[F - \Delta_{F,S}]])$. As $[[F - \Delta_{F,S}]]$ can be constructed in polynomial time and $Ver_{stage} \in$ coNP, the latter is in coNP and thus also $Ver_{stage2} \in$ coNP. The problems $Cred_{stage2}$ and $Skept_{stage2}$ can be solved by a standard guess and check algorithm, i.e. guessing an extension containing the argument (resp. not containing the argument) and using an NP-oracle to verify the extension.

For the hardness part we give a reduction $\mathcal{R}$ mapping argumentation frameworks to argumentation frameworks, such that for each AF F it holds that $stage(F) = stage2(\mathcal{R}(F))$[6]. The hardness results then follow from the corresponding hardness results for stage semantics [46].

Given an AF $F = (A, R)$ we define $\mathcal{R}(F) = (A^*, R^*)$ with $A^* = A \cup \{t\}$ and $R^* = R \cup \{(t, t)\} \cup \{(t, a), (a, t) \mid a \in A\})$, where t is a fresh argument. Then, $\mathcal{R}(F)$ consists of a

[6]Such an $\mathcal{R}$ is called an exact translation for $stage \Rightarrow stage2$ in [49].

	Ver_σ	$Cred_\sigma$	$Skept_\sigma$	NE_σ
$naive$	in P	in P	in P	in P
$stable$	in P	NP-c	coNP-c	NP-c
$cf2$	in P	NP-c	coNP-c	in P
$stage$	coNP-c	Σ_2^{P}-c	Π_2^{P}-c	in P
$stage2$	coNP-c	Σ_2^{P}-c	Π_2^{P}-c	in P

Table 5.2: Computational Complexity of naive-based semantics.

single SCC and hence $stage(\mathcal{R}(F)) = stage2(\mathcal{R}(F))$. It remains to show that $stage(F) = stage(\mathcal{R}(F))$. First, as $(t, t) \in R^*$, the argument t can not be contained in a stage extension. Furthermore, the reduction $\mathcal{R}$ does not modify attacks between arguments in A and we obtain $cf(F) = cf(\mathcal{R}(F))$. By the construction of $\mathcal{R}(F)$, for each non-empty $E \subseteq A$ we have $E_R^+ \cup \{t\} = E_{R^*}^+$ thus, $stage(F) = stage(\mathcal{R}(F))$. It is easy to see that $\emptyset \in stage(F)$ iff $cf(F) = \{\emptyset\}$ iff $\emptyset \in stage(\mathcal{R}(F))$.

The proof for $NE_{stage2} \in$ P is by the same argument as the proof of Theorem 6. □

We summarize the complexity results for naive-based semantics in Table 5.2. The results for naive semantics are due to [34], the ones for stable semantics are from [36] and the results for stage semantics have been shown in [46]. Regarding $cf2$, the complexity of $Cred_{cf2}$, $Skept_{cf2}$ and Ver_{cf2} is the same as for stable semantics, only non-emptiness is in P for $cf2$ where it is NP-complete for stable semantics. Considering the plethora of argumentation semantics, beside $stage2$, only for stage and semi-stable semantics the complexity of both skeptical and credulous reasoning is located on the second level of the polynomial hierarchy. Remember, for preferred semantics only skeptical acceptance is located on the second level of the polynomial hierarchy while credulous acceptance is NP-complete [41]. This indicates that $stage2$ is among the computationally hardest semantics but in the same breath also among the most expressive ones.

As mentioned before, the complexity results discussed so far are worst-case scenarios, for specific classes of problem instances one can achieve better results. In the next section we investigate for $cf2$ and $stage2$ semantics some possible instances where better results can be obtained.

5.3 Tractable Fragments for $cf2$ and $stage2$

As already mentioned, both $cf2$ and $stage2$ semantics are computationally intractable, i.e. the former is on the NP-layer while the latter is even on the second level of the polynomial hierarchy, naturally the issue of identifying tractable instances arises. The study of special instances of AFs where efficient algorithms can solve the reasoning problems has been done in [34, 39] as well as in [47]. In the following we study tractable fragments, i.e. classes of problem instances that can be decided in (deterministic) polynomial time.

First, we identify a relation between credulous and skeptical acceptance. By the following result, whenever credulous acceptance is tractable we immediately get tractability for skeptical acceptance.

Proposition 10. *Given an AF $F = (A, R)$ and $a \in A$ such that $(a, a) \notin R$. Then, a is skeptically accepted with $cf2$ (resp. $stage2$) iff no $\{b \mid (b, a) \in R$ or $(a, b) \in R\}$ is credulously accepted with $cf2$ (resp. $stage2$).*

Proof. For the proof we abstract from the concrete semantics $cf2$, $stage2$ and consider an arbitrary semantics σ with $\sigma(F) \subseteq naive(F)$.
$\Rightarrow$: Consider $E \in \sigma(F)$ with $a \in E$. As $E \in cf(F)$, clearly $\{b \mid (b, a) \in R$ or $(a, b) \in R\} \cap E = \emptyset$.
$\Leftarrow$: Consider $E \in \sigma(F)$ with $\{b \mid (b, a) \in R$ or $(a, b) \in R\} \cap E = \emptyset$. As $E \in naive(F)$ and $(a, a) \notin R$ we have $a \in E$. $\square$

In the following we consider different graph classes which have been proposed as tractable fragments for abstract argumentation in the literature and study the complexity of $stage2$ and $cf2$ semantics on these graph classes.

Acyclic Argumentation Frameworks

One tractable fragment for argumentation is the class of acyclic AFs. Tractability is due to the fact that on acyclic AFs most semantics coincide with the grounded semantics [37]. This result extends to $cf2$ and $stage2$.

Theorem 8. *For acyclic AFs and $\sigma \in \{cf2, stage2\}$ the problems $Cred_\sigma$ and $Skept_\sigma$ are in* P.

Proof. We first show that, on acyclic AFs, grounded, $cf2$ and $stage2$ semantics coincide. Having a look at the SCC-recursive schema applied to acyclic AFs, then the base semantics is only applied to AFs consisting of a single argument and no attack. Thus semantics coincide if they coincide on these AFs. We have $grd(\{a\}, \emptyset) = naive(\{a\}, \emptyset) = stage(\{a\}, \emptyset) = \{\{a\}\}$ and thus the assertion follows. Now the complexity results are immediate by the fact that these problems are in P for grounded semantics. $\square$

Even-Cycle Free Argumentation Frameworks

By a result in [40], reasoning with admissible-based semantics in AFs without even-length cycles is tractable. Unsurprisingly this result does not extend to $cf2$ and $stage2$ semantics.

Theorem 9. *For AFs without even-length cycles:*

- $Cred_{cf2}$ *is* NP-*complete,*
- $Skept_{cf2}$ *is* coNP-*complete,*

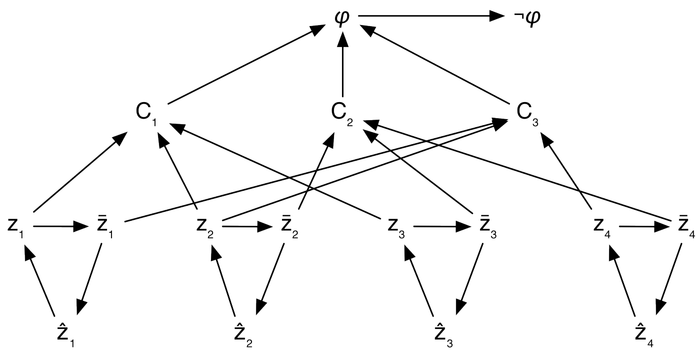

Figure 5.2: AF F_φ for the 3-CNF φ.

- Cred_{stage2} *is* NP*-hard, and*
- Skept_{stage2} *is* coNP*-hard.*

Proof. The membership part for $\mathit{cf2}$ follows immediately from the complexity results for arbitrary AFs. For the hardness part we reduce the NP-hard SAT (resp. coNP-hard UNSAT) problem to Cred (resp. Skept).

Given a 3-CNF formula $\varphi = \bigwedge_{j=1}^{m} C_j$ over atoms Z with $C_j = l_{j1} \vee l_{j2} \vee l_{j3}$ ($1 \leq j \leq m$), the corresponding AF $F_\varphi = (A_\varphi, R_\varphi)$ is built as follows:

$$\begin{array}{rcl} A_\varphi & = & Z \cup \bar{Z} \cup \hat{Z} \cup \{C_1, \ldots, C_m\} \cup \{\varphi, \neg\varphi\} \\ R_\varphi & = & \{(z, \bar{z}), (\bar{z}, \hat{z}), (\hat{z}, z) \mid z \in Z\} \cup \{(C_j, \varphi) \mid 1 \leq j \leq m\} \cup \{(\varphi, \neg\varphi)\} \cup \\ & & \{(z, C_j) \mid j \in \{1, \ldots, m\}, z \in \{l_{j1}, l_{j2}, l_{j3}\}\} \cup \\ & & \{(\bar{z}, C_j) \mid j \in \{1, \ldots, m\}, \neg z \in \{l_{j1}, l_{j2}, l_{j3}\}\} \end{array}$$

Figure 5.2 illustrates the AF F_φ of the formula $\varphi = (z_1 \vee z_2 \vee z_3) \wedge (\neg z_2 \vee \neg z_3 \vee \neg z_4) \wedge (\neg z_1 \vee z_2 \vee z_4)$.

An SCC of F_φ either consists of a single argument or is a cycle of length three which is not attacked by another SCC. As stage and naive semantics coincide on both we have $\mathit{cf2}(F_\varphi) = \mathit{stage2}(F_\varphi)$. Thus, in the remainder of the proof we only consider $\mathit{cf2}$ semantics. We now claim

(1) φ is satisfiable iff

(2) φ is credulously accepted in F_φ iff

(3) $\neg\varphi$ is not skeptically accepted in F_φ.

(1) $\Rightarrow$ (2): φ is satisfiable and thus it has a model $M \subseteq Z$. Consider the set

$$E = M \cup \{\bar{z} \mid z \in Z \setminus M\} \cup \{\varphi\}.$$

We next show, E is a *cf2* extension of F_φ. It is easy to check that $E \in naive(F_\varphi)$. So we consider $\Delta_{F_\varphi,E}$. As M is a model of φ each C_j is either attacked by a $z_i \in E$ or $\bar{z}_i \in E$, and as there are no attacks from C_j to $Z \cup \bar{Z}$ we obtain $C_j \in \Delta_{F_\varphi,E}$ for $1 \leq i \leq m$. Similarly, $\neg\varphi$ is attacked by φ, and as $\neg\varphi$ has no outgoing attacks also $\neg\varphi \in \Delta_{F_\varphi,E}$.

Now consider $Z \cup \bar{Z} \cup \hat{Z}$. Those arguments are not attacked from outside their SCCs, hence none of the arguments is contained in $\Delta_{F_\varphi,E}$. Now consider

$$F' = [[F_\varphi - \Delta_{F_\varphi,E}]] = (Z \cup \bar{Z} \cup \hat{Z} \cup \{\varphi\}, \{(z,\bar{z}), (\bar{z},\hat{z}), (\hat{z},z) \mid z \in Z\}).$$

It is easy to see that $E \in naive(F')$ and we finally obtain, $E \in cf2(F_\varphi)$. Hence, φ is credulously accepted.

(1) $\Leftarrow$ (2): Let $E \in cf2(F_\varphi)$ such that $\varphi \in E$. As E is conflict-free and $\varphi \in E$ we have $C_j \notin E$ for $1 \leq i \leq m$. Moreover $C_j \in \Delta_{F_\varphi,E}$. Assume the contrary, then there exists a $C_j \in [[F_\varphi - \Delta_{F_\varphi,E}]]$, and as C_j is not strongly connected to any argument, it is an isolated argument in the separation and thus in any naive set of $[[F_\varphi - \Delta_{F_\varphi,E}]]$, a contradiction. Now as $C_j \in \Delta_{F_\varphi,E}$, for each C_j there exists $l \in Z \cup \bar{Z}$ and $l \in E$ such that l attacks C_j (which is equivalent to $l \in C_j$). Notice, as E is conflict-free it can not happen that $\{z,\bar{z}\} \subseteq E$. Finally, we obtain $M = E \cap Z$ is a model of φ.

(2) $\Leftrightarrow$ (3): This is by the fact that in F_φ the argument $\neg\varphi$ is only connected to φ and thus each naive (resp. *cf2*) extension of F_φ either contains φ or $\neg\varphi$. $\square$

While even cycle free AFs are tractable for admissible-based semantics, in particular for stable semantics, they are still hard for *cf2*, *stage2* and also for stage semantics [54].

Bipartite Argumentation Frameworks

Bipartite AFs are a special class of frameworks where there exists a partition of the set of arguments A into two sets A_1 and A_2 such that attacks only exist between A_1 and A_2 but not within the sets.

Example 20. *Consider the AF $F = (A, R)$ as illustrated in Figure 5.3. We can partition A in $A_1 = \{a, b, d, g\}$ and $A_2 = \{c, e, f\}$, and it is easy to see that there are only attacks between those two sets. Thus, F is a bipartite argumentation framework.* $\diamond$

Bipartite AFs have been shown to be tractable for admissible based semantics [39]. In the following we show that they are also tractable for *cf2* and *stage2* semantics.

Theorem 10. *For bipartite AFs the problems $Cred_{cf2}$, $Skept_{cf2}$, Ver_{cf2} are in* P.

Proof. Given is a bipartite AF $F = (A_1, A_2, R)$ with $A = A_1 \cup A_2$. In the following we use the notation $S \rightarrowtail a$ if a set S attacks an argument a. We consider the following procedure. Start with $E_1 = A_1$ and $E_2 = \emptyset$, iterate (until E_1, E_2 reach a fixed-point)

(1) $E_2 := E_2 \cup \{b \in A_2 \mid E_1 \not\rightarrowtail b\}$ and

(2) $E_1 := E_1 \setminus \{a \in E_1 \mid E_2 \rightarrowtail a\}$.

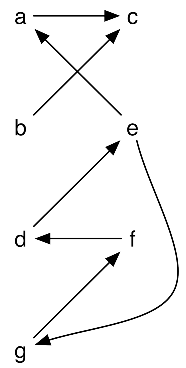

Figure 5.3: The bipartite AF F from Example 20.

By results in [39] the above algorithm works in polynomial time and computes the stable extension $S = E_1 \cup E_2$ of F, with E_1 being the set of credulously accepted arguments of F from A_1 (w.r.t. stable semantics). We next show that this algorithm also applies to *cf2*. Due to [98], in coherent systems an argument is skeptically accepted iff none of its attackers is credulously accepted. Bipartite AFs are indeed coherent, this property explains intuitively the functioning of our procedure. To this end let C_1 be the set of credulously accepted arguments of F from A_1 and S_2 the set of skeptically accepted arguments of F from A_2 (w.r.t *cf2* semantics). We claim that after each iteration step it holds that

(i) $E_1 \supseteq C_1$,

(ii) $E_2 \subseteq S_2$ and

(iii) $A_1 \setminus E_1 \subseteq \Delta_{F,S_2}$.

As an induction base observe that $E_1 = A_1$ and $E_2 = \emptyset$ trivially satisfies (i)-(iii). Now for the induction step assume (i)-(iii) holds before applying the iteration step, we have to show that it also holds afterwards.

First consider (ii): E_2 is only changed if there is a $b \in A_2$ and $E_1 \not\mapsto b$. But by (iii) this means that for all $E \in \mathit{cf2}(F)$ all attackers of b are contained in $\Delta_{F,E}$. Hence, for each $E \in \mathit{cf2}(F)$, the argument b is isolated in the AF $[[F - \Delta_{F,E}]]$ and thus clearly $b \in E$. Hence, $b \in S_2$ and (ii) is satisfied.

Now consider (i): By (2) an argument a is only removed from E_1 if it is attacked by a skeptically accepted argument. But then a can not be credulously accepted, i.e. $a \notin C_1$, and thus still $E_1 \supseteq C_1$.

Finally consider (iii): If an argument a is removed from E_1 it is attacked by an argument b such that for $E \in \mathit{cf2}(F)$ all attackers of b are contained in $\Delta_{F,E}$. Then clearly $a \not\Rightarrow_F^{A \setminus \Delta_{F,E}} b$ and thus $a \in \Delta_{F,E}$. Now using that $S = E_1 \cup E_2$ is a stable extension, the fixed-point of the above algorithm is also a *cf2* extension. Thus, $E_1 = C_1$ and $E_2 = S_2$. By symmetry we finally obtain that in bipartite AFs, the credulously (resp. skeptically) accepted arguments w.r.t. *cf2* coincide with the credulously (resp. skeptically) accepted arguments w.r.t. $stable$[7]. Hence, the P results for stable semantics in [39] carry over to $cf2$ semantics. □

In the following we illustrate the procedure of the proof of Theorem 10 on the AF of Figure 5.3.

Example 21. *Let F be the bipartite AF of Example 20 with $A_1 = \{a, b, d, g\}$ and $A_2 = \{c, e, f\}$. We start the algorithm for computing credulous and skeptical accepted arguments as in the proof above. First, for $E_1 = A_1$ and $E_2 = \emptyset$ the sets remain unchanged. Thus, we obtain $S_1 = \{a, b, d, g\}$ as a stable extension of F which is also the set of credulously accepted arguments of F from A_1, and none of the arguments from A_2 is skeptically accepted in F. Due to symmetry we consider now $E_1 = A_2$ and $E_2 = \emptyset$. Then, we obtain*

- $E_2 = \{b\}$ *and*
- $E_1 = A_2 \setminus \{c\} = \{e, f\}$.

The set $S_2 = \{b, e, f\}$ is a stable extension of F, the arguments e and f from A_2 are credulously accepted in F and $\{b\} \subset A_1$ is skeptically accepted in F (w.r.t. cf2 and stable semantics). Finally, the arguments a, b, d, g, e and f are credulously accepted in F (w.r.t. cf2 and stable semantics). ◇

Even though credulous and skeptical acceptance of $cf2$ and stable semantics coincide on bipartite AFs, they propose different extensions. For instance consider the AF F from Example 10 (illustrated on page 20). F consists of a cycle of length 6 and is a bipartite, with $A_1 = \{a, c, e\}$ and $A_2 = \{b, d, f\}$. The set $\{a, d\}$ is a $cf2$ extension of F which is not stable. Furthermore, no argument is skeptically accepted w.r.t. $cf2$ and stable semantics but all arguments are credulously accepted in F. However, for $stage2$ and stable semantics, also the extensions coincide.

Theorem 11. *For bipartite AFs $Cred_{stage2}$, $Skept_{stage2}$, Ver_{stage2} are in* P.

Proof. Bipartite AFs are odd-cycle free and therefore coherent [37]. Hence stable and stage semantics coincide. By Proposition 3 on page 40 we know that also $stable(F) = stage2(F)$. Then, the theorem follows from the results for stable semantics in [39]. □

Symmetric AFs

Finally we consider symmetric AFs, which where studied in [34]. In symmetric AFs all attacks go into both directions, hence all SCCs are isolated in the sense that there is no attack from one SCC to another (otherwise by symmetry, there would be an attack back and thus, those SCCs would merge to just one). Thus, in symmetric AFs $cf2$ coincides with naive semantics while $stage2$ coincides with stage semantics. We immediately obtain the complexity result for $cf2$ and $stage2$ by the corresponding results for naive and stage. In the first case this clearly leads to tractability. In the latter one we have to be more careful. If we follow [34] and assume that symmetric AFs are also irreflexive then, we have tractability by the fact that such AFs are

[7]By $stable(F) \subseteq stage2(F) \subseteq cf2(F)$ and Proposition 10 this also extends to $stage2$ semantics. However, this does not cover the complexity of the Ver_{stage2} problem.

	$cf2$	$stage2$	$stable$	$stage$
$Cred_\sigma{}^{acycl}$	in P	in P	P-c	P-c
$Skept_\sigma{}^{acycl}$	in P	in P	P-c	P-c
$Cred_\sigma{}^{even-free}$	NP-c	coNP-h	P-c	Σ_2^{P}-c
$Skept_\sigma{}^{even-free}$	coNP-c	coNP-h	P-c	Π_2^{P}-c
$Cred_\sigma{}^{bipart}$	in P	in P	P-c	P-c
$Skept_\sigma{}^{bipart}$	in P	in P	P-c	P-c
$Cred_\sigma{}^{sym}$	in P	in $\mathrm{P}/\Sigma_2^{\mathrm{P}}*$	in P	in P
$Skept_\sigma{}^{sym}$	in P	in $\mathrm{P}/\Pi_2^{\mathrm{P}}*$	in P	in P

Table 5.3: Complexity results for special AFs (* with self-attacking arguments).

coherent and stable semantics are tractable. However, without the assumption of irreflexivity, the tractability results for stable and stage semantics do not hold. Thus, they do not hold for $stage2$ as well.

We summarize the results for the discussed tractable fragments in Table 5.3. For comparison we also included the results for stable and stage semantics from [47].

5.4 Summary and Further Considerations

To sum up, we completed the complexity analysis for $cf2$ and $stage2$ semantics for the standard reasoning problems verification, credulous and skeptical acceptance. It turned out that both semantics are intractable, where $stage2$ is even on the second level of the polynomial hierarchy. However, deciding whether there is a non-empty extension is tractable for both semantics.

Furthermore, we considered special instances of AFs and showed that acyclic, bipartite and symmetric self-attack free frameworks are tractable for both $cf2$ and $stage2$ semantics. Whereas, if self-attacking arguments are contained in a symmetric frameworks, then we do not have tractability for $stage2$. Unsurprisingly, even-cycle free AFs are not tractable for $cf2$ and $stage2$ semantics, which reflects the special behavior of these semantics on those instances.

Another interesting approach towards tractability comes from parametrized complexity theory (see [63]). For so called fixed-parameter tractability (fpt) (see [80]), one identifies problem parameters, for instance parameters measuring the graph structure, such that computational costs heavily depend on the parameter but are only polynomial in the size of the instance. Now, if only considering problem instances with bounded parameter, one obtains a polynomial time algorithm.

First investigations for fixed-parameter tractability regarding abstract argumentation were undertaken for the graph parameters tree-width [39, 55] and clique-width [50]. The work in [56] shows that also reasoning with $cf2$ semantics is fpt w.r.t. tree-width and clique-width. Moreover, using the building blocks provided there, one can easily construct a monadic second order logic encoding for $stage2$ semantics, and by the results presented in [56] this implies fpt w.r.t. tree-width and clique-width.

Another approach towards fpt is the so called backdoor approach, using the distance to a tractable fragment as parameter [54]. In particular it was shown that the backdoor approach does not help in the case of stage semantics and as the counter examples for stage semantics immediately carry over to $stage2$ semantics[8] there is no benefit in applying the backdoor approach to $stage2$ semantics. However, in the case of $cf2$ semantics and the tractable fragments of acyclic AFs and symmetric AFs, the backdoor approach looks promising. We leave a more elaborate analysis for future work.

[8] Adding an argument that attacks itself and has a symmetric conflict with the original arguments does not change stage semantics, but ensures that stage semantics coincides with $stage2$ semantics. Indeed such an operation just increases the distance to a tractable fragment by one.

CHAPTER 6

Notions of Equivalence

Argumentation can be understood as a dynamic reasoning process, i.e. it is in particular useful to know the effects additional information causes with respect to a certain semantics. Accordingly, one can identify the information which does not contribute to the results no matter which changes are performed. In other words, we are interested in so-called *kernels* of frameworks, where two frameworks with the same kernel are then "immune" to all kind of newly added information in the sense that they always produce an equal outcome.

The concept of *strong equivalence* for argumentation frameworks captures this intuition and has been analyzed for several semantics which are all based on the concept of admissibility by Oikarinen and Woltran in [84]. Interestingly, it turned out that strong equivalence w.r.t. admissible, preferred, semi-stable and ideal semantics is exactly the same concept, while stable, complete, and grounded semantics require distinct kernels.

We complement here the picture by analyzing strong equivalence in terms of $cf2$ and $stage2$ semantics, and we compare the new results with the already existing ones. In contrast to other semantics, it turns out that for $cf2$ and $stage2$ semantics strong equivalence coincides with syntactical equivalence. We make this particular behavior more explicit by defining a new property for argumentation semantics, called the *succinctness property*. If a semantics σ satisfies the succinctness property, then for every framework F, all its attacks contribute to the evaluation of at least one framework F' containing F.

Furthermore, naive and stage have not been considered in [84] and, as they are the base semantics of $cf2$ and $stage2$ we will study them as well in this chapter. Especially in the case when an AF consists of a single SCC, the base semantics applies, thus the identification of redundant patterns for naive and stage is also relevant for our purpose. Moreover, we analyze strong equivalence for symmetric frameworks.

Strong equivalence not only gives an additional property to investigate the differences between argumentation semantics but also has some interesting applications. First, suppose we model a negotiation between two agents via argumentation frameworks. Here, strong equivalence allows to characterize situations where the two agents have an equivalent view of the world which is moreover robust to additional information.

Second, we believe that the identification of *redundant attacks* is important in choosing an appropriate semantics, in particular if an abstract argumentation framework has been built from a given knowledge base. Caminada and Amgoud outlined in [29] that the interplay between how a framework is built and which semantics is used to evaluate the framework is crucial in order to obtain useful results when the (claims of the) arguments selected by the chosen semantics are collected together. Knowledge about redundant attacks (w.r.t. a particular semantics) might help to identify unsuitable such combinations.
This chapter is organized as follows.

- In Section 6.1 we introduce the notions of standard and strong equivalence and summarize the results from the semantics studied so far in [84]. Furthermore we define the novel succinctness property for argumentation semantics.

- In Section 6.2 we first consider $cf2$ and $stage2$ semantics in terms of standard equivalence. In particular we analyze if equivalence w.r.t. a semantics implies equivalence w.r.t. another semantics. As the naive-based semantics are normally closely related to each other we also consider naive, stable and stage semantics in this context.

- Then, in Section 6.3 we first characterize strong equivalence for $cf2$ and $stage2$ semantics. Then we consider the base semantics of them namely, naive and stage.

- Finally, in Section 6.4 we compare the semantics with respect to strong equivalence and we shortly discuss strong equivalence for symmetric frameworks.

Parts of this chapter have been published in [44, 68, 69].

6.1 Background

If two distinct AFs possess the same extensions w.r.t. a semantics σ we speak about *(standard) equivalence*. Consider the following example.

Example 22. *The AFs F and G are illustrated in Figures 6.1 and 6.2. The two AFs differ in the attacks $(a, b), (a, d), (e, d), (e, b)$ and (e, c). Both AFs have no stable extension, hence $stable(F) = stable(G) = \emptyset$. Thus, F and G are equivalent with respect to stable semantics.* $\diamond$

In Section 2.2 (Figure 2.13 on page 21) we gave an overview of the relations between the semantics, and Figure 4.2 (on page 39) completes this picture for $stage2$ semantics. In the content of equivalence it is now of interest, if two AFs are equivalent w.r.t. semantics σ, are they also equivalent w.r.t. semantics τ? Oikarinen and Woltran investigated the relations between equivalence for many semantics in [84]. In the following we briefly summarize the results.

Proposition 11. *For any AFs F and G, we have*

- $adm(F) = adm(G) \Longrightarrow pref(F) = pref(G)$;

- $adm(F) = adm(G) \Longrightarrow ideal(F) = ideal(G)$;

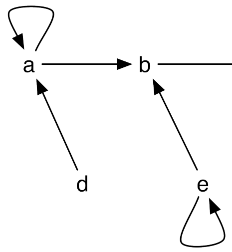

Figure 6.1: AF F from Example 22.

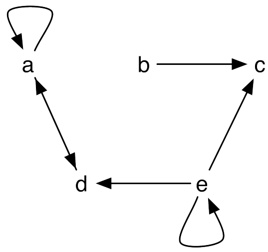

Figure 6.2: AF G from Example 22.

- $compl(F) = compl(G) \Longrightarrow pref(F) = pref(G)$;
- $compl(F) = compl(G) \Longrightarrow grd(F) = grd(G)$;
- $compl(F) = compl(G) \Longrightarrow ideal(F) = ideal(G)$.

There is no particular relation between equivalence for the remaining combinations of stable, admissible, preferred, complete, grounded, ideal, semi-stable and eager semantics.

Argumentation is a dynamic reasoning process, therefore we are interested in identifying information which does not contribute to the results no matter which changes are performed. In the next subsection we consider *strong equivalence* for AFs, a concept which reflects this intuition.

Strong Equivalence for AFs

Strong equivalence for argumentation frameworks not only requires that two AFs have the same extensions under a specific semantics but also, if the frameworks are augmented with additional information, they still possess the same extensions (under the semantics). The following example illustrates this for stable semantics.

Example 23. *Consider the AFs F and G from Example 22 (Figures 6.1 and 6.2). We add the new AF $H = (\{b, e\}, \{(b, e)\})$ to each of them. Then, they still have the same stable extensions $stable(F \cup H) = stable(G \cup H) = \{\{b, d\}\}$, as highlighted in the graphs of Figures 6.3 and 6.4. Furthermore, it can be shown that no matter which framework H one adds to F and G they will always posses the same stable extensions.* $\diamond$

The concept of strong equivalence for argumentation frameworks, as introduced by Oikarinen and Woltran in [84], meets exactly the behavior described in Example 23. The formal definition is as follows.

Definition 30. *Two AFs F and G are* strongly equivalent *to each other w.r.t. a semantics σ, in symbols $F \equiv_s^\sigma G$, iff for each AF H, $\sigma(F \cup H) = \sigma(G \cup H)$.*

By definition, $F \equiv_s^\sigma G$ implies $\sigma(F) = \sigma(G)$, but the other direction is not true in general. To characterize strong equivalence, Oikarinen and Woltran used in [84] so-called *kernels* for

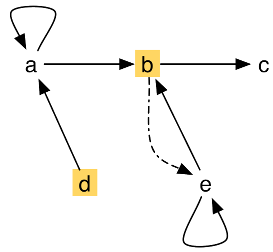

Figure 6.3: $F \cup H$ from Example 23.

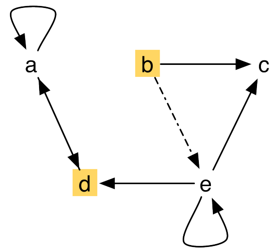

Figure 6.4: $G \cup H$ from Example 23.

different semantics which implicitly remove the redundant attacks of the compared frameworks. As shown in [84], deciding strong equivalence then amounts to checking the syntactic equivalence of the kernels of the two compared frameworks. More precisely, such kernels have been provided for many semantics, viz. for admissible, preferred, ideal, semi-stable, eager, complete and grounded semantics. All these kernels are non-trivial in the sense that certain attacks are removed.

In the following we recapitulate the respective kernels for the semantics considered in [84].

Definition 31. *For an AF $F = (A, R)$, we define*

- $R^{sk} = R \setminus \{(a, b) \mid a \neq b, (a, a) \in R\}$*, and* $F^{sk} = (A, R^{sk})$ *as the* s-kernel *of* F*;*
- $R^{ak} = R \setminus \{(a, b) \mid a \neq b, (a, a) \in R, \{(b, a), (b, b)\} \cap R \neq \emptyset\}$*, and* $F^{ak} = (A, R^{ak})$ *as the* a-kernel *of* F*;*
- $R^{gk} = R \setminus \{(a, b) \mid a \neq b, (b, b) \in R, \{(a, a), (b, a)\} \cap R \neq \emptyset\}$*, and* $F^{gk} = (A, R^{gk})$ *as the* g-kernel *of* F*;*
- $R^{ck} = R \setminus \{(a, b) \mid a \neq b, (a, a), (b, b) \in R\}$*, and* $F^{ck} = (A, R^{ck})$ *as the* c-kernel *of* F*.*

The next proposition summarizes the results obtained in [84].

Proposition 12. *For any AFs F and G:*

- $F^{sk} = G^{sk}$ *iff* $F \equiv_s^{stable} G$*;*
- $F^{ak} = G^{ak}$ *iff* $F \equiv_s^{\sigma} G$*, where* $\sigma \in \{adm, semis, pref, ideal, eager\}$*;*
- $F^{gk} = G^{ak}$ *iff* $F \equiv_s^{grd} G$*;*
- $F^{ck} = G^{ck}$ *iff* $F \equiv_s^{compl} G$*.*

Inspecting the respective kernels provides the following picture, for any AFs F, G:

$$F = G \Rightarrow F^{ck} = G^{ck} \Rightarrow F^{ak} = G^{ak} \Rightarrow F^{sk} = G^{sk}; \quad F^{ck} = G^{ck} \Rightarrow F^{gk} = G^{gk} \tag{6.1}$$

and thus, strong equivalence w.r.t. complete semantics implies strong equivalence w.r.t. grounded semantics as well as strong equivalence w.r.t. admissible sets (and thus w.r.t. preferred, ideal, and semi-stable semantics); finally, strong equivalence w.r.t. admissible sets implies strong equivalence w.r.t. stable semantics.

The Succinctness Property

When considering strong equivalence for argumentation frameworks it turns out that for most semantics there can be identified redundant attacks. Hence, there exists some information in those frameworks which has no influence on the extensions, i.e. there is at least one attack in one of the frameworks which can be removed without changing the extensions. Thus, this attack is *redundant* w.r.t. semantics σ.

In the next definition we make this idea formal; for AFs $F = (A, R)$ and $F' = (A', R')$ we write $F \subseteq F'$ to denote that $A \subseteq A'$ and $R \subseteq R'$ jointly hold. Moreover, we use $F \setminus (a, b)$ as a shorthand for the framework $(A, R \setminus \{(a, b)\})$.

Definition 32. *For an AF $F = (A, R)$ and semantics σ we call an attack $(a, b) \in R$* redundant *in F* w.r.t. *σ iff for all F' with $F \subseteq F'$, $\sigma(F') = \sigma(F' \setminus (a, b))$.*

Consider the AFs of Example 23. There, the attacks $\{(a, b), (e, b)\}$ in F as well as the attacks $\{(a, d), (e, c), (e, d)\}$ in G are redundant under stable semantics.

However, in the context of strong equivalence one compares particular frameworks, here we define a general property for argumentation semantics. With the *succinctness property* we are able to evaluate semantics independent of the specific instantiation method. Therefore, the succinctness property can be seen as an additional criterion for the evaluation of argumentation semantics, similar to the one proposed by Baroni and Giacomin [8].

The succinctness property identifies to which extend attacks contribute in terms of a given semantics. In other words, we are interested here in how many attacks are possibly ignored in the computation of a semantics. The concept of succinctness is now captured as follows.

Definition 33. *An argumentation semantics σ satisfies the* succinctness property *or is* maximal succinct *iff no AF contains a redundant attack w.r.t. σ.*

The following theorem gives the link between the succinctness property and strong equivalence.

Theorem 12. *An argumentation semantics σ satisfies the succinctness property iff for any AFs F and G, strong equivalence between F and G w.r.t. σ coincides with syntactic equivalence, i.e. $F = G$.*

Proof. Suppose σ does not satisfy the the succinctness property, i.e. there exists an F and an attack (a, b) in F such that $\sigma(F \cup H) = \sigma((F \setminus (a, b)) \cup H)$ for any AF H. Obviously, $F \equiv_s^\sigma F \setminus (a, b)$ but $F \neq F \setminus (a, b)$.

Suppose $F \neq G$ but $F \equiv_\sigma^s G$. W.l.o.g. let (a, b) be an attack in F which does not occur in G. Since $F \equiv_\sigma^s G$, $\sigma(F \cup H) = \sigma(G \cup H)$, in particular for all H not containing (a, b). Since $F \cup H \cup (a, b) = F \cup H$, we get that $\sigma(G \cup (a, b) \cup H) = \sigma(G \cup H)$ for all H. By setting $G' = G \cup (a, b)$, we observe that (a, b) is redundant in G' w.r.t. σ. Hence, σ cannot be maximal succinct. $\square$

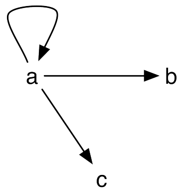

Figure 6.5: AF F from Example 24.

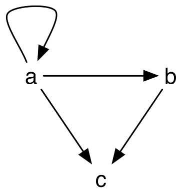

Figure 6.6: AF G from Example 24.

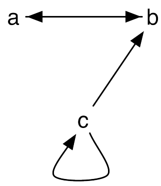

Figure 6.7: AF F from Example 24.

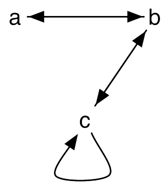

Figure 6.8: AF G from Example 24.

6.2 Standard Equivalence

In this section we take a closer look at the relations between $cf2$ and $stage2$ semantics and the other naive-based ones in terms of equivalence. Especially we are interested if equivalence w.r.t. a semantics implies equivalence w.r.t. another semantics? Normally the relations between $cf2$ and the other semantics in terms of subset inclusion is as depicted in Figure 4.2 (on page 39). Here we only consider the naive-based semantics $cf2$, $stage2$, stable, stage and naive in more detail, because for the admissible-based semantics we already have no relation w.r.t. subset inclusion (as one can observe in Figure 2.13 on page 21).

In particular if odd-length cycles are involved in the frameworks there is no relation between $cf2$ and admissible-based semantics in terms of equivalence. The next example shows that in general for two AFs F and G $adm(F) = adm(G) \not\Longrightarrow cf2(F) = cf2(G)$ and $cf2(F) = cf2(G) \not\Longrightarrow adm(F) = adm(G)$.

Example 24. *Consider the AFs F and G as illustrated in Figures 6.5 and 6.6. We have $adm(F) = adm(G) = \{\emptyset\}$ but $cf2(F) = \{\{b, c\}\} \neq cf2(G) = \{\{b\}\}$. For the other direction, let F and G be as in Figures 6.7 and 6.8. Then, $cf2(F) = cf2(G) = \{\{a\}, \{b\}\}$ but $adm(F) = \{\emptyset, \{a\}\} \neq adm(G) = \{\emptyset, \{a\}, \{b\}\}$.* $\diamond$

In the following we concentrate on the relations between $cf2$ and the other naive-based semantics and we show in the next examples that there is no particular relation between naive, stage, stable, $cf2$ and $stage2$ semantics in terms of standard equivalence which means that two frameworks possess the same extensions under a given semantics.

First, we consider AFs F and G such that $\sigma(F) = \sigma(G) \not\Longrightarrow \theta(F) = \theta(G)$, where $\sigma \in \{naive, stage, stable\}$ and $\theta \in \{cf2, stage2\}$.

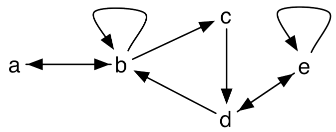

Figure 6.9: AF F from Example 25.

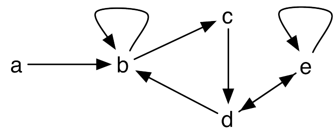

Figure 6.10: AF G from Example 25.

Example 25. *Let F and G be as illustrated in Figures 6.9 and 6.10. The only difference between those two AFs is the attack (b, a) which is contained in F but not in G. This has the effect that the framework F consists of a single SCC; and thus $cf2(F) = naive(F)$ and $stage2(F) = stage(F)$. We have $stable(F) = stable(G) = \emptyset$ and $stage(F) = stage(G) = \{\{a, c\}, \{a, d\}\}$. Furthermore, $naive(F) = naive(G) = \{\{a, c\}, \{a, d\}\}$. However, we have $cf2(F) = stage2(F) = \{\{a, c\}, \{a, d\}\}$ and $cf2(G) = stage2(G) = \{\{a, c\}\}$.*

On the other hand, $S = \{a, d\}$ is not a cf2 extension of G, since $\Delta_{G,S} = \{b\}$,

$$[[G - \Delta_{G,S}]] = (\{a, c, d, e\}, \{(d, e), (e, d), (e, e)\}),$$

and thus $naive([[G - \Delta_{G,S}]]) = \{\{a, c, d\}\}$. For stage2 and the set S we observe $stage([[G - \Delta_{G,S}]]) = \{\{a, c, d\}\}$, and thus S is no stage2 extension of G. Hence,

$$\sigma(F) = \sigma(G) \not\Longrightarrow \theta(F) = \theta(G)$$

for $\sigma \in \{naive, stage, stable\}$, and $\theta \in \{cf2, stage2\}$ as desired. $\diamond$

The next example shows that $\sigma(F) = \sigma(G) \not\Longrightarrow \theta(F) = \theta(G)$, where $\sigma \in \{naive, cf2, stage2\}$ and $\theta \in \{stage, stable\}$.

Example 26. *The AFs F and G are illustrated in Figures 6.11 and 6.12. Then, we obtain*

- $naive(F) = naive(G) = \{\{a\}, \{b\}\}$,
- $cf2(F) = cf2(G) = \{\{a\}\}$ *and*
- $stage2(F) = stage2(G) = \{\{a\}\}$.

On the other side

- $stable(F) = \emptyset \neq stable(G) = \{\{a\}\}$ *and*
- $stage(F) = \{\{a\}, \{b\}\} \neq stage(G) = \{\{a\}\}$.

Thus, we showed that $\sigma(F) = \sigma(G) \not\Longrightarrow \theta(F) = \theta(G)$, for $\sigma \in \{naive, cf2, stage2\}$ and $\theta \in \{stage, stable\}$. $\diamond$

Now, we provide frameworks F and G such that $\sigma(F) = \sigma(G) \not\Longrightarrow naive(F) = naive(G)$, where $\sigma \in \{stable, stage, cf2, stage2\}$.

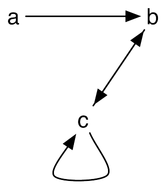

Figure 6.11: AF F from Example 26.

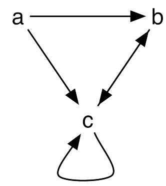

Figure 6.12: AF G from Example 26.

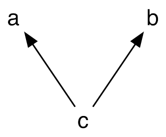

Figure 6.13: AF F from Example 27.

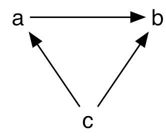

Figure 6.14: AF G from Example 27.

Example 27. *Let the AFs F and G be as in Figures 6.13 and 6.14. Then, we have $\sigma(F) = \sigma(G) = \{\{c\}\}$, where $\sigma \in \{stable, stage, cf2, stage2\}$ but $naive(F) = \{\{a,b\},\{c\}\}$ and $naive(G) = \{\{a\},\{b\},\{c\}\}$.* $\diamond$

Finally, we look at some AFs such that $stable(F) = stable(G) \not\Longrightarrow stage(F) = stage(G)$ and $stage(F) = stage(G) \not\Longrightarrow stable(F) = stable(G)$.

Example 28. *Let the AFs F, G and H be as follows:*

- $F = (\{a,b\},\{(a,a),(b,b)\})$,
- $G = (\{a,b\},\{(b,b)\})$,
- $H = (\{a,b\},\{(a,b),(b,b)\})$.

Then, $stable(F) = stable(G) = \emptyset$ but $stage(F) = \{\emptyset\} \neq \{\{a\}\} = stage(G)$; and $stage(G) = stage(H) = \{\{a\}\}$ but $stable(G) = \emptyset \neq \{\{a\}\} = stable(H)$. $\diamond$

6.3 Strong Equivalence

In what follows, we characterize strong equivalence for $cf2$ and $stage2$ semantics as well as for their base semantics naive and stage. All of them have not been considered in [84]. As it turns out, for $cf2$ and $stage2$ semantics strong equivalence amounts to syntactic equivalence, which means that both of them satisfy the succinctness property. On the other hand the characterizations for naive and stage semantics do not coincide with syntactical equivalence, thus they are not maximal succinct.

In the following we provide three lemmata which will be useful later. The first shows that in case two frameworks do not posses the same arguments one can always extend them in a way that they do not coincide w.r.t. naive, stage, $cf2$ and $stage2$ semantics.

Lemma 9. *For any AFs F and G with $A(F) \neq A(G)$, there exists an AF H such that $A(H) \subseteq A(F) \cup A(G)$ and $\sigma(F \cup H) \neq \sigma(G \cup H)$, for the semantics $\sigma \in \{naive, stage, cf2, stage2\}$.*

Proof. In case $\sigma(F) \neq \sigma(G)$, we just consider $H = (\emptyset, \emptyset)$ and get $\sigma(F \cup H) \neq \sigma(G \cup H)$. Thus assume $\sigma(F) = \sigma(G)$ and let w.l.o.g. $a \in A(F) \setminus A(G)$. Thus for all $E \in \sigma(F)$, $a \notin E$. Consider the framework $H = (\{a\}, \emptyset)$. Then, for all $E' \in \sigma(G \cup H)$, we have $a \in E'$. On the other hand, $F \cup H = F$ and also $\sigma(F \cup H) = \sigma(F)$. Hence, a is not contained in any $E \in \sigma(F \cup H)$, and we obtain $\sigma(F \cup H) \neq \sigma(G \cup H)$. $\square$

The next lemma states that two frameworks at least need to coincide with regard to self-attacking arguments.

Lemma 10. *For any AFs F and G such that $(a, a) \in R(F) \setminus R(G)$ or $(a, a) \in R(G) \setminus R(F)$, there exists an AF H such that $A(H) \subseteq A(F) \cup A(G)$ and $\sigma(F \cup H) \neq \sigma(G \cup H)$, for $\sigma \in \{naive, stage, cf2, stage2\}$.*

Proof. Let the self-attack $(a, a) \in R(F) \setminus R(G)$ and consider the framework

$$H = (A, \{(a, b), (b, b) \mid a \neq b \in A\})$$

with $A = A(F) \cup A(G)$. Then $\sigma(G \cup H) = \{a\}$ while $\sigma(F \cup H) = \{\emptyset\}$ for all considered semantics $\sigma \in \{naive, stage, cf2, stage2\}$. For example, in case $\sigma = cf2$ we obtain $\Delta_{G \cup H, E} = \{b \mid b \in A \setminus \{a\}\}$. Moreover, $\{a\}$ is conflict-free in $G \cup H$ and $\{a\} \in naive(G')$, where $G' = (G \cup H) - \Delta_{G \cup H, E} = (\{a\}, \emptyset)$. On the other hand, $cf2(F \cup H) = \{\emptyset\}$ since all arguments in $F \cup H$ are self-attacking. The case for $(a, a) \in R(G) \setminus R(F)$ is similar. $\square$

The following lemma shows that if a set S is conflict-free in the union of two AFs then the intersection of S with the arguments of each of the two AFs is also conflict-free in the single AFs (and the other way around).

Lemma 11. *Let F and H be AFs and S be a set of arguments. Then, $S \in cf(F \cup H)$ iff, jointly $(S \cap A(F)) \in cf(F)$ and $(S \cap A(H)) \in cf(H)$.*

Proof. The only-if direction is clear. Thus suppose $S \notin cf(F \cup H)$. Then, there exist $a, b \in S$, such that $(a, b) \in F \cup H$. By our definition of "$\cup$", then $(a, b) \in F$ or $(a, b) \in H$. But then $(S \cap A(F)) \notin cf(F)$ or $(S \cap A(H)) \notin cf(H)$ follows. $\square$

In the next subsection we start our analysis of strong equivalence with the $cf2$ semantics.

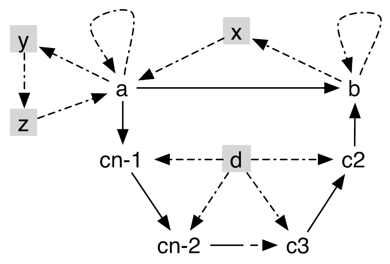

Figure 6.15: $F \cup H$.

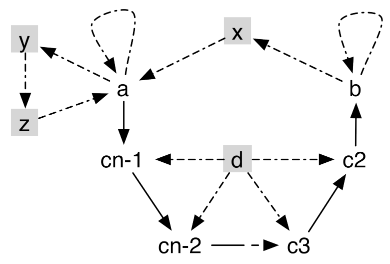

Figure 6.16: $G \cup H$.

Strong Equivalence w.r.t. *cf2* Semantics

Interestingly, it turns out that for this semantics there are no redundant attacks at all. In fact, even in the case where an attack links two self-attacking arguments, this attack might play a role by gluing two components together. Having no redundant attacks means that strong equivalence coincides with syntactic equivalence.

Theorem 13. *For any AFs F and G, $F \equiv_s^{cf2} G$ iff $F = G$.*

Proof. Since for any AFs $F = G$ obviously implies for all AFs H, $cf2(F \cup H) = cf2(G \cup H)$, we only have to show that if $F \neq G$ there exists an AF H such that $cf2(F \cup H) \neq cf2(G \cup H)$. From Lemma 9 and Lemma 10 we know that in case the arguments or the self-loops are not equal in both frameworks, there exists an AF H such that $cf2(F \cup H) \neq cf2(G \cup H)$. We thus assume that $A = A(F) = A(G)$ and $(a, a) \in R(F)$ iff $(a, a) \in R(G)$, for each $a \in A$. Let us thus suppose w.l.o.g. an attack $(a, b) \in R(F) \setminus R(G)$ and consider the AF

$$\begin{aligned} H &= (A \cup \{d, x, y, z\}, \{(a, a), (b, b), (b, x), (x, a), (a, y), (y, z), (z, a), \\ &\quad (d, c) \mid c \in A \setminus \{a, b\}\}), \end{aligned}$$

see also Figures 6.15 and 6.16 for illustration. Then, for a set $E = \{d, x, z\}$, we have $E \in cf2(F \cup H)$ but $E \notin cf2(G \cup H)$.

To show that $E \in cf2(F \cup H)$, we first compute $\Delta_{F \cup H,E} = \{c \mid c \in A \setminus \{a, b\}\}$. Thus, we have two SCCs left in the instance $[[(F \cup H) - \Delta_{F \cup H,E}]]$, namely $C_1 = \{d\}$ and $C_2 = \{a, b, x, y, z\}$ as illustrated in Figure 6.17. Furthermore, all attacks between the arguments of C_2 are preserved, and we obtain that $E \in naive([[(F \cup H) - \Delta_{F \cup H,E}]])$, and as $E \in naive(F \cup H)$ holds, we have that $E \in cf2(F \cup H)$ as well.

On the other hand, we obtain $\Delta_{G \cup H,E} = \{a\} \cup \{c \mid c \in A \setminus \{a, b\}\}$, and the instance $G' = [[(G \cup H) - \Delta_{G \cup H,E}]]$ consisting of five SCCs, namely $C_1 = \{d\}$, $C_2 = \{b\}$, $C_3 = \{x\}$, $C_4 = \{y\}$ and $C_5 = \{z\}$, with b being self-attacking as illustrated in Figure 6.18. Thus, the set $E' = \{d, x, y, z\} \supset E$ is conflict-free in G'. Therefore, we obtain $E \notin naive(G')$, and hence, $E \notin cf2(G \cup H)$. $F \not\equiv_s^{cf2} G$ follows. $\square$

In other words, the proof of Theorem 13 shows that no matter which AFs $F \neq G$ are given, one can always construct a framework H such that $cf2(F \cup H) \neq cf2(G \cup H)$. In particular, we can

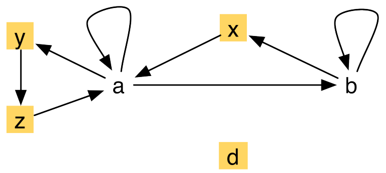

Figure 6.17: $[[(F \cup H) - \Delta_{F \cup H, E}]]$.

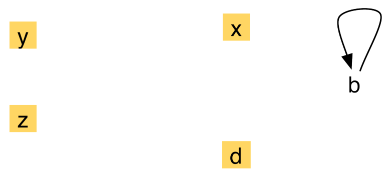

Figure 6.18: $[[(G \cup H) - \Delta_{G \cup H, E}]]$.

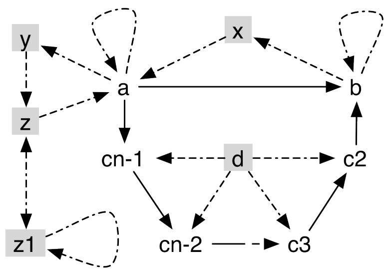

Figure 6.19: $F \cup H$.

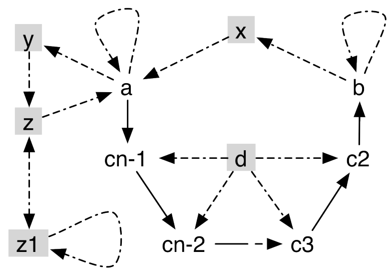

Figure 6.20: $G \cup H$.

always add new arguments and attacks such that the missing attack in one of the original frameworks leads to different SCCs in the modified ones and therefore to different *cf2* extensions, when suitably augmenting the two AFs under comparison.

This special behavior of *cf2* leads us to the next observation that *cf2* is the first semantics considered so far, which is maximal succinct. By Theorem 12 and Theorem 13 the following result is obvious.

Corollary 1. *The cf2 semantics satisfies the succinctness property.*

Strong Equivalence w.r.t. *stage2* Semantics

In the previous subsection we showed that for *cf2* semantics, strong equivalence coincides with syntactic equivalence. In other words, there are no redundant patterns at all. In the following, we show that the same holds for *stage2* semantics.

Theorem 14. *For any AFs F and G, $F \equiv_s^{stage2} G$ iff $F = G$.*

Proof. Since for any AFs $F = G$ obviously implies for all AFs H, $stage2(F \cup H) = stage2(G \cup H)$, we only have to show that if $F \neq G$ there exists an AF H such that $stage2(F \cup H) \neq stage2(G \cup H)$.

For any two AFs F and G, strong equivalence w.r.t. naive-based semantics requires that the AFs coincide with the arguments and the self-attacks (Lemma 9 and Lemma 10). We thus assume that $A = A(F) = A(G)$ and $(a, a) \in R(F)$ iff $(a, a) \in R(G)$, for each $a \in A$. Let us thus

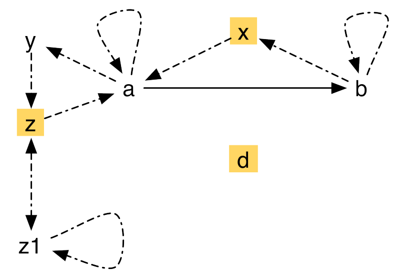

Figure 6.21: $[[(F \cup H) - \Delta_{F \cup H,E}]]$.

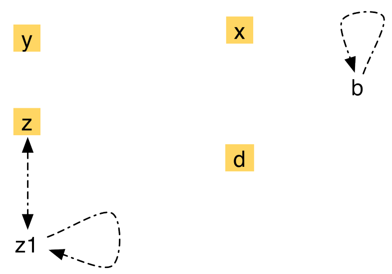

Figure 6.22: $[[(G \cup H) - \Delta_{G \cup H,E}]]$.

suppose w.l.o.g. an attack $(a, b) \in R(F) \setminus R(G)$ and consider the AF

$$\begin{aligned} H \quad = \quad & (A \cup \{d, x, y, z, z1\}, \{(a, a), (b, b), (b, x), (x, a), (a, y), (y, z), (z, a), \\ & (z, z1), (z1, z), (z1, z1), (d, c) \mid c \in A \setminus \{a, b\}\}), \end{aligned}$$

see also Figures 6.19 and 6.20 for illustration.

Then, for $E = \{d, x, z\}$, we have $E \in \mathit{stage2}(F \cup H)$ but $E \notin \mathit{stage2}(G \cup H)$. To show that $E \in \mathit{stage2}(F \cup H)$, we first compute $\Delta_{F \cup H,E} = \{c \mid c \in A \setminus \{a, b\}\}$. Thus, we have two SCCs left in the instance $F' = [[(F \cup H) - \Delta_{F \cup H,E}]]$, namely $C_1 = \{d\}$ and $C_2 = \{a, b, x, y, z, z1\}$ as illustrated in Figure 6.21. Furthermore, all attacks between the arguments of C_2 are preserved, and we obtain that $E \in \mathit{stage}(F')$, and as $E \in \mathit{naive}(F \cup H)$, $E \in \mathit{stage2}(F \cup H)$ follows.

On the other hand, we obtain $\Delta_{G \cup H,E} = \{a\} \cup \{c \mid c \in A \setminus \{a, b\}\}$, and the instance $G' = [[(G \cup H) - \Delta_{G \cup H,E}]]$ consists of five SCCs, namely $C_1 = \{d\}$, $C_2 = \{b\}$, $C_3 = \{x\}$, $C_4 = \{y\}$ and $C_5 = \{z, z1\}$, with b and $z1$ being self-attacking as illustrated in Figure 6.22.

Thus, the set $T = \{d, x, y, z\} \supset E$ is conflict-free in G' and $T^+_{R(G')} \supset E^+_{R(G')}$. Therefore, we obtain $E \notin \mathit{stage}(G')$, and hence, $E \notin \mathit{stage2}(G \cup H)$. $F \not\equiv^{stage2}_s G$ follows. $\square$

By Theorem 12 and Theorem 14 the following result is obvious.

Corollary 2. *The $\mathit{stage2}$ semantics satisfies the succinctness property.*

The *cf2* and *stage2* semantics are the only semantics considered so far, where strong equivalence coincides with syntactic equivalence. This can be seen as another special property of them which is met by the succinctness property.

We continue our investigation with the two base semantics of *cf2* and *stage2*, namely the naive and stage semantics.

Strong Equivalence w.r.t. Naive Semantics

For naive semantics, strong equivalence is only a marginally more restricted concept than standard equivalence, namely in case the two compared AFs are not given over the same arguments.

Theorem 15. *For any AFs F and G, the following statements are equivalent:*

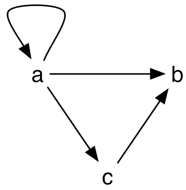

Figure 6.23: AF F from Example 29.

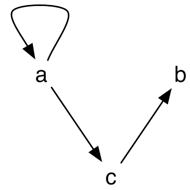

Figure 6.24: AF G from Example 29.

(1) $F \equiv_s^{naive} G$;

(2) $naive(F) = naive(G)$ *and* $A(F) = A(G)$;

(3) $cf(F) = cf(G)$ *and* $A(F) = A(G)$.

Proof. (1) implies (2): basically by the definition of strong equivalence and Lemma 9.

(2) implies (3): Assume $naive(F) = naive(G)$ but $cf(F) \neq cf(G)$. W.l.o.g. let $S \in cf(F) \setminus cf(G)$. Then, there exists a set $S' \supseteq S$ such that $S' \in naive(F)$ and by assumption then $S' \in naive(G)$. However, as $S \notin cf(G)$ there exists an attack $(a, b) \in R(G)$, such that $a, b \in S$. But as $S \subseteq S'$, we have $S' \notin cf(G)$ as well; a contradiction to $S' \in naive(G)$.

(3) implies (1): Suppose $F \not\equiv_s^{naive} G$, i.e. there exists a framework H such that $naive(F \cup H) \neq naive(G \cup H)$. W.l.o.g. let now $S \in naive(F \cup H) \setminus naive(G \cup H)$. From Lemma 11 one can show that $(S \cap A(F)) \in naive(F)$ and $(S \cap A(H)) \in naive(H)$, as well as $(S \cap A(G)) \notin naive(G)$. Let us assume $S' = S \cap A(F) = S \cap A(G)$, otherwise we are done yielding $A(F) \neq A(G)$. If $S' \notin cf(G)$ we are also done (since $S' \in cf(F)$ follows from $S' \in naive(F)$); otherwise, there exists an $S'' \supset S'$, such that $S'' \in cf(G)$. But $S'' \notin cf(F)$, since $S' \in naive(F)$. Again we obtain $cf(F) \neq cf(G)$ which concludes the proof. □

By Theorem 12 and Theorem 15 we obtain the next result.

Corollary 3. *The naive semantics is not maximal succinct.*

Strong Equivalence w.r.t. Stage Semantics

In order to characterize strong equivalence w.r.t. stage semantics, we require here exactly the same kernel as already used in [84] to characterize strong equivalence w.r.t. stable semantics.

Example 29. *Consider the frameworks F and G as illustrated in Figures 6.23 and 6.24. They only differ in the attacks outgoing from the argument a which is self-attacking and yield the same single stage extension, namely $\{c\}$, for both frameworks. We can now add, for instance, $H = (\{a, c\}, \{(c, a)\})$ and the stage extensions for $F \cup H$ and $G \cup H$ still remain the same. In fact, no matter how H looks like, $stage(F \cup H) = stage(G \cup H)$ will hold.* ◇

The s-kernel from Definition 31 reflects the intuition given in the previous example. The following theorem states that two AFs are strongly equivalent with respect to stage semantics if they have the same s-kernel.

Theorem 16. *For any AFs F and G, $F \equiv_s^{stage} G$ iff $F^{sk} = G^{sk}$.*

Proof. Only-if: Suppose $F^{sk} \neq G^{sk}$, we show that $F \not\equiv_s^{stage} G$. From Lemma 9 and Lemma 10 we know that in case the arguments or the self-loops are not equal in both frameworks, $F \equiv_s^{stage} G$ does not hold. We thus assume that $A = A(F) = A(G)$ and $(a, a) \in F$ iff $(a, a) \in G$, for each $a \in A$. Let thus w.l.o.g. $(a, b) \in F^{sk} \setminus G^{sk}$. We can conclude $(a, b) \in F$ and $(a, a) \notin F$, thus $(a, a) \notin G$ and $(a, b) \notin G$. Let c be a fresh argument and take

$$H = \{A \cup \{c\}, \{(b, b)\} \cup \{(c, d) \mid d \in A\} \cup \{(a, d) \mid d \in A \cup \{c\} \setminus \{b\}\}).$$

Then, $\{a\}$ is a stage extension of $F \cup H$ (it attacks all other arguments) but not of $G \cup H$ (b is not attacked by $\{a\}$); see also Figures 6.25 and 6.26 for illustration.

For the if-direction, suppose $F^{sk} = G^{sk}$. Let us first show that $F^{sk} = G^{sk}$ implies $cf(F \cup H) = cf(G \cup H)$, for each AF H. Towards a contradiction, suppose an H such that $cf(F \cup H) \neq cf(G \cup H)$ and w.l.o.g. let $T \in cf(F \cup H) \setminus cf(G \cup H)$. Since $F^{sk} = G^{sk}$, we know $A(F) = A(G)$. Thus there exist $a, b \in T$ (not necessarily $a \neq b$) such that $(a, b) \in G \cup H$ or $(b, a) \in G \cup H$. On the other hand $(a, b) \notin F \cup H$ and $(b, a) \notin F \cup H$ hold since $a, b \in T$ and $T \in cf(F \cup H)$. Thus, in particular, $(a, b) \notin F$ and $(b, a) \notin F$ as well as $(a, b) \notin H$ and $(b, a) \notin H$; due to Lemma 11 the latter implies $(a, b) \in G$ or $(b, a) \in G$. Suppose $(a, b) \in G$ (the other case is symmetric). If $(a, a) \in G$ then $(a, a) \in G^{sk}$, but $(a, a) \notin F^{sk}$ (since $a \in T$ and thus $(a, a) \notin F$). If $(a, a) \notin G$, $(a, b) \in G^{sk}$ but $(a, b) \notin F^{sk}$ (since $(a, b) \notin F$). In either case $F^{sk} \neq G^{sk}$, a contradiction.

We next show that $F^{sk} = G^{sk}$ implies $(F \cup H)^{sk} = (G \cup H)^{sk}$ for any AF H. Thus, let $(a, b) \in (F \cup H)^{sk}$, and assume $F^{sk} = G^{sk}$; we show $(a, b) \in (G \cup H)^{sk}$. Since, $(a, b) \in (F \cup H)^{sk}$ we know that $(a, a) \notin F \cup H$ and therefore, $(a, a) \notin F^{sk}$, $(a, a) \notin G^{sk}$ and $(a, a) \notin H^{sk}$. Hence, we have either $(a, b) \in F^{sk}$ or $(a, b) \in H^{sk}$. In the later case, $(a, b) \in (G \cup H)^{sk}$ follows because $(a, a) \notin G^{sk}$ and $(a, a) \notin H^{sk}$. In case $(a, b) \in F^{sk}$, we get by the assumption $F^{sk} = G^{sk}$, that $(a, b) \in G^{sk}$ and since $(a, a) \notin H^{sk}$ it follows that $(a, b) \in (G \cup H)^{sk}$.

Finally we show that for any frameworks K and L such that $K^{sk} = L^{sk}$, and any $S \in cf(K) \cap cf(L)$, $S_R^+(K) = S_R^+(L)$. This follows from the fact that for each $s \in S$, (s, s) is neither contained in K nor in L. But then each attack $(s, b) \in K$ is also in K^{sk}, and likewise, each attack $(s, b) \in L$ is also in L^{sk}. Now since $K^{sk} = L^{sk}$, $S_R^+(K) = S_R^+(L)$ is obvious.

Thus, we showed that, given $F^{sk} = G^{sk}$, the following relations hold for each AF H:

- $cf(F \cup H) = cf(G \cup H)$;
- $(F \cup H)^{sk} = (G \cup H)^{sk}$; and
- $S_R^+(F \cup H) = S_R^+(G \cup H)$ holds, for each $S \in cf(F \cup H) = cf(G \cup H)$ (taking $K = F \cup H$ and $L = G \cup H$).

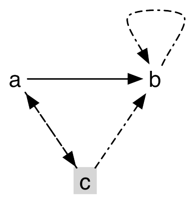

Figure 6.25: $F \cup H$.

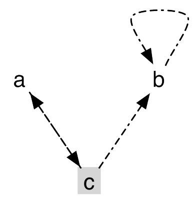

Figure 6.26: $G \cup H$.

Thus, $stage(F \cup H) = stage(G \cup H)$, for each AF H. Consequently, $F \equiv_s^{stage} G$. $\square$

From Theorem 16 and Proposition 12 we obtain that strong equivalence for stable and stage semantics coincide. By Theorem 12 and Theorem 16 we obtain the next result.

Corollary 4. *The stage semantics is not maximal succinct.*

Recall that the results in [84] in combination with Theorem 12 show that many other semantics are not maximal succinct.

6.4 Discussion and Further Considerations

In this section, we first compare our new results to the known results from [84] in order to get a complete picture about the difference between the most important semantics in terms of strong equivalence and redundant attacks. Afterwards, we restrict ourselves to symmetric AFs [34]. This is motivated by the fact that naive-based semantics do not take the orientation of attacks into account.

Comparing Semantics w.r.t. Strong Equivalence

Together with the results from [84], we now know how to characterize strong equivalence for the following semantics of abstract argumentation: admissible, preferred, complete, grounded, stable, semi-stable, ideal, stage, naive, *cf2* and $stage2$. The first five semantics (which are due to Dung [37]) as well as semi-stable [25] and ideal [38] semantics yield as extensions admissible sets. The later four semantics do not yield admissible sets in general. Nonetheless, thanks to our characterizations we get now a clear picture which kind of attacks are redundant w.r.t. a certain semantics. First of all, the kernel we used for stage semantics (see Definition 31) exactly matches the kernel for stable semantics in [84]. We thus get:

Corollary 5. *For any AFs F and G, $F \equiv_s^{stable} G$ holds iff $F \equiv_s^{stage} G$ holds.*

According to (6.1) on page 65 we conclude that strong equivalence w.r.t. *cf2* semantics implies strong equivalence w.r.t. complete semantics, etc. To complete the picture, we also note the following observation:

Lemma 12. *If $F^{sk} = G^{sk}$ (resp. $F^{gk} = G^{gk}$), then $cf(F) = cf(G)$.*

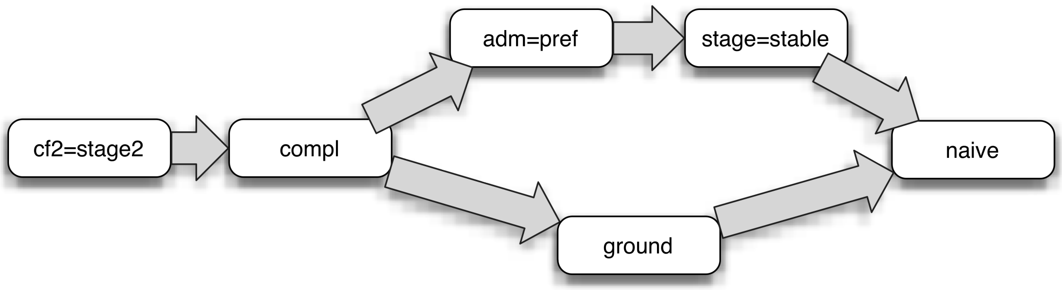

Figure 6.27: Full picture of implication in terms of strong equivalence.

Proof. If $F^{sk} = G^{sk}$ then due to Lemma 9, $A = A(F) = A(G)$ and from Lemma 10 we know that for each $a \in A$, $(a,a) \in R(F)$ iff $(a,a) \in R(G)$. Let $S \in cf(F)$, i.e. for each $a, b \in S$, we have $(a,b) \notin R(F)$. Then, $(a,b) \notin R(F^{sk})$ and by assumption $(a,b) \notin R(G^{sk})$. Now since $a \in S$, we know that $(a,a) \notin R(F)$ and thus $(a,a) \notin R(G)$. Then, $(a,b) \notin R(G^{sk})$ implies $(a,b) \notin R(G)$. Since this is the case for any $a, b \in S$, $S \in cf(G)$ follows. The converse direction is analogous.
Showing that $F^{gk} = G^{gk}$ implies $cf(F) = cf(G)$ can be done by similar arguments. □

As an immediate consequence of the above lemma and Theorem 15, we obtain

Corollary 6. *For any AFs F and G, we have that $F \equiv_s^\sigma G \Rightarrow F \equiv_s^{naive} G$ (for $\sigma \in \{stable, stage, grd\}$).*

Together with our previous observation we thus obtain a complete picture of implications in terms of strong equivalence w.r.t. to the different semantics as depicted in Figure 6.27.

Inspecting the notions of kernels, we also observe that in the case when self-loop free AFs are compared, all notions of strong equivalence except the one of naive semantics coincide.

Corollary 7. *Strong equivalence between self-loop free AFs F and G w.r.t. admissible, preferred, complete, grounded, stable, semi-stable, ideal, stage, $cf2$ and $stage2$ semantics holds, if and only if $F = G$.*

For naive semantics, we might have situations where $F \equiv_s^{naive} G$ holds although F and G are different self-loop free AFs. As a simple example consider $F = (\{a,b\}, \{(a,b)\})$ and $G = (\{a,b\}, \{(b,a)\})$. As already mentioned earlier, this is due to the fact that naive semantics do not take the orientation of attacks into account. This motivates to compare semantics w.r.t. strong equivalence for symmetric frameworks.

Strong Equivalence and Symmetric Frameworks

Symmetric frameworks have been studied in [34] and are defined as AFs $F = (A, R)$ where R is symmetric, non-empty, and irreflexive. We consider here a more relaxed such notion. We call an AF (A, R) *weakly symmetric* if R is symmetric (but not necessarily non-empty or irreflexive).

Strong equivalence between weakly symmetric AFs is defined analogously as in Definition 30, i.e. weakly symmetric AFs F and G are strongly equivalent w.r.t. a semantics σ iff $\sigma(F \cup H) = \sigma(G \cup H)$, for any AF H. Note that we do not restrict here that H is symmetric as well.

For *cf2* and *stage2* semantics, strong equivalence between weakly symmetric AFs still requires $F = G$ (basically, this follows from the fact that all steps in the proof of Theorem 13 can be restricted to such frameworks). Regarding the other semantics, we have two main observations. First, we can now give a suitable realization for the concept of a kernel also in terms of naive semantics.

Definition 34. *For an AF $F = (A, R)$, define $F^{nk} = (A, R^{nk})$ where*

$$R^{nk} = R \setminus \{(a, b) \mid a \neq b, (a, a) \in R \textit{ or } (b, b) \in R\}.$$

Theorem 17. *For any weakly symmetric AFs F and G, $F \equiv_s^{naive} G$ iff $F^{nk} = G^{nk}$.*

Proof. By Theorem 15, it is sufficient to show that $F^{nk} = G^{nk}$ holds iff jointly $A(F) = A(G)$ and $cf(F) = cf(G)$. Obviously, $F^{nk} = G^{nk}$ implies $A(F) = A(G)$. Thus, let $S \in cf(F)$. Then, for each $a, b \in S$, neither (a, a) nor (b, b) is contained in $R(F)$. Furthermore, we have $\{(a, b), (b, a\} \cap R(F) = \emptyset$. Thus, we obtain $\{(a, a), (b, b), (a, b), (b, a)\} \cap R(F^{nk}) = \emptyset$. By the assumption $F^{nk} = G^{nk}$, we know $\{(a, a), (b, b), (a, b), (b, a)\} \cap R(G^{nk}) = \emptyset$, and thus neither (a, a) nor (b, b) is contained in $R(G)$. But then, $\{(a, b), (b, a)\} \cap R(G) = \emptyset$; hence there is no conflict between a and b in G as well. Since this holds for all pairs $a, b \in S$, we get $S \in cf(G)$. The other direction is analogous.

Thus, suppose $F^{nk} \neq G^{nk}$. In case, $A(F^{nk}) \neq A(G^{nk})$ (i.e. $A(F) \neq A(G)$) we can employ Lemma 9. In case, there exists an a such that (a, a) is contained in exactly one, $R(F)$ or $R(G)$, we employ Lemma 10. In both cases we obtain $F \not\equiv_s^{naive} G$. Thus, assume F and G possess the same self-loops. Since $F^{nk} \neq G^{nk}$, there exist distinct arguments a, b such that w.l.o.g. $(a, b) \in R(F^{nk}) \setminus R(G^{nk})$. Since, $(a, b) \in R(F^{nk})$, $\{(a, a), (b, b)\} \cap R(F) = \emptyset$ and by our assumption above, also $\{(a, a), (b, b)\} \cap R(G) = \emptyset$, thus $(a, b) \notin R(G)$. Moreover, since G is weakly symmetric, also $(b, a) \notin R(G)$. It follows, $\{a, b\} \in cf(G)$ but $\{a, b\} \notin cf(F)$. By Theorem 15, $F \not\equiv_s^{naive} G$. □

Second, one can show that for any weakly symmetric AF F, it holds that $F^{sk} = F^{ak}$. This leads to the following result.

Corollary 8. *Strong equivalence between weakly symmetric AFs F and G w.r.t. admissible, preferred, semi-stable, ideal, stable, and stage semantics coincides.*

Furthermore, we can simplify the kernel F^{gk} for weakly symmetric AFs to $F^{gk} = (A, R \setminus \{(a, b) \mid a \neq b, (b, b) \in R\})$. The other kernels remain unchanged for weakly symmetric AFs.

Finally, we note here that in case the augmented AF is symmetric as well, which means that none of the AFs contain self-loops, it follows from Corollary 7 that symmetric strong equivalence between two AFs F and G w.r.t. semantics admissible, preferred, complete, grounded, stable, semi-stable, ideal, stage, *cf2* and *stage2* semantics holds, if and only if $F = G$.

6.5 Conclusion

In this chapter we provided characterizations for strong equivalence w.r.t. $cf2$ and $stage2$ semantics as well as for their base semantics stage and naive. Thus, we completed the analysis initiated in [84]. Strong equivalence gives a handle to identify redundant attacks.

The identification of redundant attacks is an important preprocessing step and can help to simplify frameworks before the evaluation. In particular, the knowledge about redundancies can already been taken into account during the instantiation process. The newly introduced succinctness property is then satisfied by a semantics if for every framework F, all its attacks contribute to the evaluation (i.e. all attacks are non-redundant) of at least one framework F' containing F.

Redundant attacks exist for all semantics (at least when self-loops are present), except for $cf2$ and $stage2$ semantics, which follows from our main result, that $F \equiv_s^{cf2} G$ (resp. $F \equiv_s^{stage2} G$) holds, if and only if, $F = G$. In other words, each attack plays a role for these two semantics (at least, an attack closes a cycle and thus is crucial for the actual partition into SCCs of the AF). Thus, $cf2$ and $stage2$ semantics are maximal succinct. Our result also strengthens the observations from Baroni *et al.* [12], who claim that $cf2$ semantics treats self-loops in a more sensitive way than other semantics.

Regarding stage and naive, we showed that strong equivalence w.r.t. stage semantics coincides with strong equivalence w.r.t. stable semantics. For naive semantics it is only required that the AFs are given over the same arguments and have the same conflict-free sets. From the obtained results we conclude that none of the prominent semantics, except $cf2$ and $stage2$, satisfies the succinctness property. Besides our characterization for strong equivalence, we also analyzed symmetric strong equivalence.

CHAPTER 7

Implementation

In this chapter we concentrate on the more practical part of our investigation. In order to evaluate and compare abstract argumentation frameworks with respect to the numerous semantics it is indispensable to have efficient systems. As already pointed out in Chapter 5, argumentation problems are in general intractable. Therefore, developing dedicated algorithms for the different reasoning problems is non-trivial. A promising way to implement such systems is to use a *reduction method*, where the given problem is translated into another language, for which sophisticated systems already exist. It turned out that *Answer-Set Programming* (ASP) is well suited for this purpose due to the following three characteristics.

- The prototypical language of ASP (i.e., logic programming under the answer-set semantics [72], also known as stable logic programming or A-Prolog) is very expressible and allows to formulate queries (in an extended datalog fashion) over databases, such that multiple results can be obtained. In our context, queries thus can be employed to obtain multiple extensions for AFs, where the actual AF to process is just given as an input database.
- Advanced ASP-solvers such as clasp, claspD, DLV, Cmodels, Smodels, IDP, or SUP are nowadays able to deal with large problem instances, see, e.g., [24]. Thus, using our proposed reduction method delegates the burden of optimization to these systems.
- Depending on the syntactical structure of a given ASP query, the complexity of evaluating that query on input databases (i.e., the data complexity of ASP) varies from classes P, NP, coNP up to Σ_2^{P} and to Π_2^{P}. Hence, for the different types of problems in abstract argumentation, we are able to formulate queries which are "well suited" from a complexity point of view. In other words, the complexity of evaluating ASP queries representing some argumentation problem lies in the same complexity class as the original problem.

Many argumentation semantics have been already implemented in ASP, see [95] for an overview. In this work we follow the ASPARTIX approach [57, 59, 66], where a single program is used to encode a particular semantics, while the instance of the framework is given as an input database. In particular we will present ASP encodings for $cf2$ and $stage2$ semantics.

The challenging part in the design of the encodings for $cf2$ (resp. $stage2$) semantics is that the original definition involves a recursive computation of different sub-frameworks which is rather cumbersome to represent directly in ASP. This was the main reason why we invented the alternative characterization for $cf2$ semantics as presented in Chapter 3 (resp. Chapter 4 for $stage2$ semantics). With the novel characterization we are able to directly (i) guess a set S of the given AF F and then (ii) check whether S is a naive (resp. stage) extension of the instance $[[F - \Delta_{F,S}]]$. While the encodings for $cf2$ are quite short and comprehensible this is not the case for the standard encodings for $stage2$ semantics. This semantics is located on the second level of the polynomial hierarchy and is based on stage semantics which requires a test for subset-maximality. To perform this test we need to apply a certain *saturation technique* [60] which is hardly accessible for non-experts in ASP.

However, recent advances in ASP solvers, in particular, the `metasp` optimization front-end for the ASP-system `gringo/claspD` allows for much simpler encodings for such tests. More precisely, `metasp` allows to use the traditional $\#minimize$ statement (which in its standard variant minimizes w.r.t. cardinality or weights, but not w.r.t. subset inclusion) also for selection among answer sets which are minimal w.r.t. subset inclusion in certain predicates. Details about `metasp` can be found in [70]. We will use this optimization to simplify the encodings for $stage2$ (resp. stage) semantics.

Besides the ASP approach we will consider the *labeling-based approach* as a direct implementation method. Lately algorithms based on labelings attracted specific attention [27, 28, 79, 83, 97]. In contrast to the traditional extension-based approach, so called labelings (see e.g. [14]) distinguish two kinds of unaccepted arguments, those which are rejected by the extension and those which are neither rejected nor accepted. This distinction is interesting from a logic perspective but has also proven to be useful for algorithmic issues. We will present two algorithms which compute all valid labelings for $cf2$ and $stage2$ semantics.

As third contribution we sketch the web-application of the system ASPARTIX[9]. This is a user friendly tool which allows to use ASPARTIX without the need of downloading or installing any ASP solver or encodings. The platform is directly accessible form the web with any standard browser and provides a graphical representation of the input framework and the solutions.

The remainder of this chapter is organized as follows: in Section 7.1 we provide the necessary background on ASP, then we give the encodings for $cf2$ semantics. Before we present the encodings of $stage2$ semantics we explain the saturation encodings for the base semantics of $stage2$, namely the stage semantics, followed by the `metasp` encodings for stage semantics. Then we provide first the saturation and then the `metasp` encodings for $stage2$ semantics. In Section 7.2 we give the definitions of $cf2$ and $stage2$ semantics followed by two algorithms which compute all labelings of these semantics. Finally we close the chapter with a short discussion of the results.

Parts of this chapter have been published in [45, 52, 59, 66, 67].

[9] See `http://rull.dbai.tuwien.ac.at:8080/ASPARTIX/`.

7.1 ASP-Encodings for Abstract Argumentation Frameworks

In this section we consider ASP as a reduction-based approach for the implementation of AFs. First we introduce the necessary background on ASP, then we formalize how argumentation frameworks are represented in ASP and we give the encodings for *cf2* and *stage2* semantics. All our encodings are fixed where the instance of an AF is given as input, and they are incorporated in the system ASPARTIX (see [57, 59] for more details) and available online[10]. The encodings from the system ASPARTIX are written in the general datalog syntax. It may be the case that one needs to adapt the encodings for some ASP solvers. The `metasp` encodings can only be performed with `gringo/claspD`. Furthermore we point out that in this section we give an informal description of the ASP encodings. For a more formal investigation we refer to [59].

Background Answer-Set Programming

We first give a brief overview of the syntax and semantics of the ASP-formalism we consider, i.e., disjunctive datalog under the answer-sets semantics [72]. As we will use the `metasp` optimization front-end for for the ASP-system `gringo/claspD` to simplify the encodings for stage semantics, we also briefly introduce the syntax and semantics of the $\#minimize$ statement (which in its standard variant minimizes w.r.t. cardinality or weights, but not w.r.t. subset inclusion). The `metasp` front-end allows to use the $\#minimize$ statement also for selection among answer-sets which are minimal w.r.t. subset inclusion in certain predicates. Details about `metasp` can be found in [70]. Finally, we briefly recall some important complexity results for disjunctive datalog. We refer to [61, 70, 71, 74] for a broader exposition on all of these topics.

In what follows, we fix a countable set $\mathcal{U}$ of *(domain) elements*, also called *constants* and suppose a total order $<$ over these elements. An *atom* is an expression $p(t_1, \ldots, t_n)$, where p is a *predicate* symbol of arity $n \geq 0$ and each t_i is either a variable or an element from $\mathcal{U}$. An atom is *ground* if it is free of variables.

A *(disjunctive) rule* r is of the form

$$a_1 \lor \cdots \lor a_n \leftarrow b_1, \ldots, b_k, \ not\ b_{k+1}, \ldots, \ not\ b_m$$

with $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$, and where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms, and "not " stands for *default negation*. The *head* of r is the set $H(r) = \{a_1, \ldots, a_n\}$ and the *body* of r is $B(r) = \{b_1, \ldots, b_k,\ not\ b_{k+1}, \ldots,\ not\ b_m\}$. Furthermore, $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$. A rule r is *normal* (or *disjunction-free*) if $n \leq 1$ and a *constraint* if $n = 0$. A rule r is *safe* if each variable in r occurs in $B^+(r)$. A rule r is *ground* if no variable occurs in r. If each rule in a program is normal (resp., ground), we call the program normal (resp., ground). A *fact* is a disjunction-free ground rule with an empty body.

A program is a finite set of safe (disjunctive) rules. Employing database notation, we call a finite set of facts also an *input database* and a set of non-ground rules a *query*. For a program π and an input database D, we often write $\pi(D)$, instead of the program $D \cup \pi$, in order to indicate that D serves as input for π.

[10] `http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage/`

A normal program π is called *stratified* if no atom a depends by recursion through negation on itself [4]. More formally, π is stratified if there exists an assignment $\alpha(\cdot)$ of integers to the predicates in π, such that for each rule $r \in \pi$, the following holds: If predicate p occurs in the head of r and predicate q occurs

(i) in the positive body of r, then $\alpha(p) \geq \alpha(q)$ holds;

(ii) in the negative body of r, then $\alpha(p) > \alpha(q)$ holds.

As an example, consider the following program π

$$\begin{aligned}\pi = \{\ & a(X) \leftarrow \mathit{not}\ b(X), d(X);\\ & b(X) \leftarrow a(X)\ \}.\end{aligned}$$

In order to find an assignment $\alpha(\cdot)$ satisfying the above conditions for π, observe that the first rule of π requires $\alpha(a) > \alpha(b)$, but the second rule, in turn, forces $\alpha(b) \geq \alpha(a)$. In other words, each assignment $\alpha(\cdot)$ violates at least one of the conditions, and hence, π is not stratified. For the following program

$$\begin{aligned}\pi' = \{\ & a(X) \leftarrow \mathit{not}\ b(X), d(X);\\ & b(X) \leftarrow c(X);\\ & c(X) \leftarrow b(X)\ \},\end{aligned}$$

we can use the assignment $\alpha(a) = 2, \alpha(b) = \alpha(c) = \alpha(d) = 1$ to show that π' is stratified. The concept of stratified programs is very important in logic programming, since it allows for a restricted form of negation, but does not lead to an increase in the complexity (see also the complexity results below, which show that stratified programs still can be evaluated efficiently, while this is not the case for normal or disjunctive programs).

Besides disjunctive and normal programs, we consider here the class of optimization programs, i.e. normal programs which additionally contain $\#\mathit{minimize}$ statements

$$\#\mathit{minimize}[l_1 = w_1 @ J_1, \ldots, l_k = w_k @ J_k] \tag{7.1}$$

where l_i is a literal, w_i an integer weight and J_i an integer priority level.

For any program π, let $\mathcal{U}_\pi$ be the set of all constants appearing in π (if no constant appears in π, an arbitrary constant is added to $\mathcal{U}_\pi$), and let B_π be the set of all ground atoms constructible from the predicate symbols appearing in π and the constants of $\mathcal{U}_\pi$. Moreover, $Gr(\pi)$ is the set of rules $r\tau$ obtained by applying, to each rule $r \in \pi$, all possible substitutions τ from the variables in π to elements of $\mathcal{U}_\pi$.

Let the set of all ground atoms over $\mathcal{U}$ be denoted by $B_\mathcal{U}$. An *interpretation* $I \subseteq B_\mathcal{U}$ *satisfies* a ground rule r iff $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. A ground program π is satisfied by I, if I satisfies each $r \in \pi$. A non-ground rule r (resp., a non-ground program π) is satisfied by I, if I satisfies all groundings of r (resp., $Gr(\pi)$). An interpretation $I \subseteq B_\mathcal{U}$ is an *answer-set* of π iff it is a subset-minimal set satisfying the *Gelfond-Lifschitz reduct*

$$\pi^I = \{\ H(r) \leftarrow B^+(r) \mid I \cap B^-(r) = \emptyset, r \in Gr(\pi)\ \}.$$

	stratified programs	normal programs	disjunctive programs	`metasp` programs
$\models_c$	P	NP	Σ_2^{P}	Σ_2^{P}
$\models_s$	P	coNP	Π_2^{P}	Π_2^{P}

Table 7.1: Data Complexity for datalog (all results are completeness results).

For a program π, we denote the set of its answer-sets by $\mathcal{AS}(\pi)$. We note that for each $I \in \mathcal{AS}(\pi)$, $I \subseteq B_\pi$ holds. Moreover, a program can have multiple answer-sets. A stratified program has at most one answer-set, and a constraint-free stratified program has exactly one answer-set.

For semantics of optimization programs, we interpret the $\#minimize$ statement w.r.t. subset-inclusion: For any sets X and Y of atoms, we have $Y \subseteq_J^w X$, if for any weighted literal $l = w@J$ occurring in (7.1), $Y \models l$ implies $X \models l$. Then, M is a collection of relations of the form $\subseteq_J^w$ for priority levels J and weights w. A standard answer-set (i.e. not taking the minimize statements into account) Y of π *dominates* a standard answer-set X of π w.r.t. M if there are a priority level J and a weight w such that $X \subseteq_J^w Y$ does not hold for $\subseteq_J^w \in M$, while $Y \subseteq_{J'}^{w'} X$ holds for all $\subseteq_{J'}^{w'} \in M$ where $J' \geq J$. Finally a standard answer-set X is an answer-set of an optimization program π w.r.t. M if there is no standard answer-set Y of π that dominates X w.r.t. M.

We briefly recall some central complexity results for ASP. Credulous and skeptical reasoning in terms of programs are defined as follows. Given a program π and a set A of ground atoms. We denote by $\pi \models_c A$ that A is contained in some answer-sets of π. Likewise, we denote by $\pi \models_s A$ that A is contained in all answer-sets of π. In the former case, we reason *credulously*, in the latter case, we reason *skeptically*. Since we will deal with fixed programs, we focus on results for data complexity. Recall that data complexity in our context addresses the problem $\pi(D) \models A$ where the query π is fixed, while the input database D and ground atoms A are inputs of the decision problem. Depending on the concrete definition of $\models$, we get the complexity results in Table 7.1, compiled from [35] and the references therein.

Representing AFs in ASP

Here we first show how to represent AFs in ASP, and we give two programs which we need later on in this section. The first one π_{cf} opens the search space for our solutions via two guessing rules and eliminates all guesses which are not conflict-free. The second program $\pi_<$ defines an order over the domain elements.

All our programs are fixed which means that the only translation required, is to give an AF F as input database $\hat{F}$ to the program π_σ for a semantics σ. In fact, for an AF $F = (A, R)$, we define $\hat{F}$ as

$$\hat{F} = \{ \operatorname{arg}(a) \mid a \in A\} \cup \{\operatorname{att}(a, b) \mid (a, b) \in R \}.$$

In what follows, we use unary predicates $\mathrm{in}/1$ and $\mathrm{out}/1$ to perform a guess for a set $S \subseteq A$, where $\mathrm{in}(a)$ represents that $a \in S$ (resp. $\mathrm{out}(a)$ for $a \notin S$). The following notion of correspondence is relevant for our purposes.

Definition 35. *Let $\mathcal{S} \subseteq 2^{\mathcal{U}}$ be a collection of sets of domain elements and let $\mathcal{I} \subseteq 2^{B_{\mathcal{U}}}$ be a collection of sets of ground atoms. We say that $\mathcal{S}$ and $\mathcal{I}$ correspond to each other, in symbols $\mathcal{S} \cong \mathcal{I}$, iff*

(i) for each $S \in \mathcal{S}$, there exists an $I \in \mathcal{I}$, such that $\{a \mid \mathrm{in}(a) \in I\} = S$;

(ii) for each $I \in \mathcal{I}$, it holds that $\{a \mid \mathrm{in}(a) \in I\} \in \mathcal{S}$; and

(iii) $|\mathcal{S}| = |\mathcal{I}|$.

Let $F = (A, R)$ be an AF. The following program fragment guesses, when augmented by $\hat{F}$, any subset $S \subseteq A$ and then checks whether the guess is conflict-free in F:

$$\begin{aligned} \pi_{cf} = \{\ & \mathrm{in}(X) \leftarrow \mathit{not}\ \mathrm{out}(X), \mathrm{arg}(X); \\ & \mathrm{out}(X) \leftarrow \mathit{not}\ \mathrm{in}(X), \mathrm{arg}(X); \\ & \leftarrow \mathrm{in}(X), \mathrm{in}(Y), \mathrm{att}(X, Y)\ \}. \end{aligned}$$

Proposition 13. *For any AF F, $\mathit{cf}(F) \cong \mathcal{AS}(\pi_{cf}(\hat{F}))$.*

The proof of Proposition 13 can be found in [59]. For ASP encodings, it is sometimes required or desired to avoid the use of negation. This might either be the case for the saturation technique or if a simple program can be solved without a Guess&Check approach. Then, encodings typically rely on a form of loops where all domain elements are visited and it is checked whether a desired property holds for all elements visited so far. We will use this technique in our saturation-based encoding in the upcoming subsection, but also for the computation of the instance $[[F - \Delta_{F,S}]]$ for $\mathit{cf2}$ and $\mathit{stage2}$ semantics.

For this purpose, an order $<$ over the domain elements (usually provided by common ASP solvers) is used together with a few helper predicates defined in program $\pi_<$ below; in fact, predicates $\inf/1, \mathrm{succ}/2$ and $\sup/1$ denote infimum, successor and supremum of the order $<$.

$$\begin{aligned} \pi_< = \{\ & \mathrm{lt}(X, Y) \leftarrow \mathrm{arg}(X), \mathrm{arg}(Y), X < Y; \\ & \mathrm{nsucc}(X, Z) \leftarrow \mathrm{lt}(X, Y), \mathrm{lt}(Y, Z); \\ & \mathrm{succ}(X, Y) \leftarrow \mathrm{lt}(X, Y), \mathit{not}\ \mathrm{nsucc}(X, Y); \\ & \mathrm{ninf}(Y) \leftarrow \mathrm{lt}(X, Y); \\ & \inf(X) \leftarrow \mathrm{arg}(X), \mathit{not}\ \mathrm{ninf}(X); \\ & \mathrm{nsup}(X) \leftarrow \mathrm{lt}(X, Y); \\ & \sup(X) \leftarrow \mathrm{arg}(X), \mathit{not}\ \mathrm{nsup}(X)\ \}. \end{aligned}$$

ASP-Encodings for $\mathit{cf2}$ Semantics

To this end, we provide a fixed program π_{cf2} which, augmented with an input database representing a given AF F, has its answer-sets in a one-to-one correspondence to the $\mathit{cf2}$ extensions of F. In particular, π_{cf2} computes $\mathit{cf2}$ extension along the lines of Theorem 1. The modularity of ASP allows us to split π_{cf2} into several modules, where we also make use of the two program moduls π_{cf} and $\pi_{<}$ introduced above. Then, the program π_{cf2} implements the following steps, given an AF $F = (A, R)$:

1. *Guess* the conflict-free sets $S \subseteq A$ of F.
2. For each S, compute the set $\Delta_{F,S}$.
3. For each S, derive the *instance* $[[F - \Delta_{F,S}]]$.
4. *Check* whether $S \in naive([[F - \Delta_{F,S}]])$.

Step 1 is computed by π_{cf}, thus we go directly to Step 2. In the module π_{reach} we use the predicates $\inf(\cdot)$, $\mathrm{succ}(\cdot, \cdot)$ and $\sup(\cdot)$ from the module $\pi_{<}$ to iterate over the operator $\Delta_{F,S}(\cdot)$. Given $F = (A, R)$, by definition of $\Delta_{F,S}$ it is sufficient to compute at most $|A|$ such iterations to reach the fixed-point. Let us now present the module and then explain its behavior in more detail.

$$
\begin{aligned}
\pi_{reach} = \{\ & \mathrm{arg\_set}(N, X) \leftarrow \mathrm{arg}(X), \inf(N); && (7.2)\\
& \mathrm{reach}(N, X, Y) \leftarrow \mathrm{arg\_set}(N, X), \mathrm{arg\_set}(N, Y), \mathrm{att}(X, Y); && (7.3)\\
& \mathrm{reach}(N, X, Y) \leftarrow \mathrm{arg\_set}(N, X), \mathrm{att}(X, Z), \mathrm{reach}(N, Z, Y); && (7.4)\\
& \mathrm{d}(N, X) \leftarrow \mathrm{arg\_set}(N, Y), \mathrm{arg\_set}(N, X), \mathrm{in}(Y), \mathrm{att}(Y, X),\\
& \qquad\qquad\quad \mathit{not}\ \mathrm{reach}(N, X, Y); && (7.5)\\
& \mathrm{arg\_set}(M, X) \leftarrow \mathrm{arg\_set}(N, X), \mathit{not}\ \mathrm{d}(N, X), \mathrm{succ}(N, M)\ \}. && (7.6)
\end{aligned}
$$

Rule (7.2) first copies all arguments into a set indexed by the infimum which initiates the computation. The remaining rules are applicable to arbitrary indices, whereby rule (7.6) copies (a subset of the) arguments from the currently computed set into the "next" set using the successor function $\mathrm{succ}(\cdot, \cdot)$. This guarantees a step-by-step computation of $\mathrm{arg\_set}(i, \cdot)$ by incrementing the index i. The functioning of rules (7.3)–(7.6) is as follows. Rules (7.3) and (7.4) compute a predicate $\mathrm{reach}(n, x, y)$ indicating that there is a path from argument x to argument y in the given framework *restricted* to the arguments of the current set n. In rule (7.5), $\mathrm{d}(n, x)$ is obtained for all arguments x which are component-defeated by S in this restricted framework. In other words, if n is the i-th argument in the order $<$, $\mathrm{d}(n, x)$ carries exactly those arguments x which are contained in $\Delta^{i}_{F,S}$. Finally, rule (7.6) copies arguments from the current set which are *not* component-defeated to the successor set.

Next, we derive the instance $[[F - \Delta_{F,S}]]$ with the module π_{inst}. As already outlined above, if the supremum m is reached in π_{reach}, we are guaranteed that the derived atoms $\mathrm{arg\_set}(m, x)$ characterize exactly those arguments x from the given AF F which are not contained in $\Delta_{F,S}$. It is thus now relatively easy to obtain the instance $[[F - \Delta_{F,S}]]$ which is done below via predicates $\mathrm{arg\_new}(\cdot)$ and $\mathrm{att\_new}(\cdot, \cdot)$.

$$
\begin{aligned}
\pi_{inst} = \{\ & \mathrm{arg\_new}(X) \leftarrow \mathrm{arg\_set}(M, X), \sup(M);\\
& \mathrm{att\_new}(X, Y) \leftarrow \mathrm{arg\_new}(X), \mathrm{arg\_new}(Y), \mathrm{att}(X, Y),\\
& \qquad\qquad\qquad\qquad \mathrm{reach}(M, Y, X), \sup(M)\ \}.
\end{aligned}
$$

Finally, it remains to verify whether the initially guessed set S is a *cf2* extension. To do so, we need to check whether $S \in naive([[F - \Delta_{F,S}]])$. The following module does this job by checking whether only those arguments are not contained in S, for which an addition to S would yield a conflict.

$$
\begin{aligned}
\pi_{check\_naive} = \{\ & \mathrm{conflicting}(X) \leftarrow \mathrm{att\_new}(Y, X), \mathrm{out}(X), \mathrm{in}(Y);\\
& \mathrm{conflicting}(X) \leftarrow \mathrm{att\_new}(X, Y), \mathrm{out}(X), \mathrm{in}(Y);\\
& \mathrm{conflicting}(X) \leftarrow \mathrm{att\_new}(X, X);\\
& \leftarrow \mathit{not}\ \mathrm{conflicting}(X), \mathrm{out}(X), \mathrm{arg\_new}(X)\ \}.
\end{aligned}
$$

One important observation here is that the checking module has no influence on the guessing part. This will not be the case when we come to the encodings for $stage2$ (resp. stage) semantics. We now have our entire encoding available:

$$\pi_{cf2} = \pi_{cf} \cup \pi_{<} \cup \pi_{reach} \cup \pi_{inst} \cup \pi_{check\_naive}.$$

The desired correspondence between answer-sets and *cf2* extensions is as follows.

Proposition 14. *For any AF F, $cf2(F) \cong \mathcal{AS}(\pi_{cf2}(\hat{F}))$.*

ASP-Encodings for $stage2$ Semantics

Here we concentrate on implementing the $stage2$ semantics in ASP. Therefore we will first give the encodings for stage semantics, and then due to the modularity of ASP we can reuse those encodings for the ones for $stage2$ semantics. In the previous subsection we saw that the encodings for *cf2* semantics are quite simple and short, this can be explained by the low complexity of the individual components. For stage and $stage2$ semantics we need more involved programming techniques like saturation encodings which we will explain in the following.

Saturation Encodings for Stage Semantics

We exemplify on the stage semantics the saturation technique for encodings which solve associated problems which are on the second level of the polynomial hierarchy. This technique was introduced by Eiter and Gottlob in [60] and it was already used to encode the preferred and semi-stable semantics in [59]. While with default negation, one is capable to formulate an exclusive guess (as we did in the encodings for *cf2* semantics), disjunction can be employed for the saturation technique, which allows for representing even more complex problems. The term “saturation” indicates that all atoms which are subject to a guess can also be jointly contained in an interpretation. To saturate a guess, it is however necessary that the checking part of a program interacts with the guessing part.

The encodings for the stage semantics are very similar to the one of semi-stable extensions from [59]. The main difference is that for semi-stable extensions the set $S \subseteq A$ needs to be admissible, whereas for stage extensions the set S is only required to be conflict-free. Therefore we obtain the encoding for stage extensions by a slight modification of the encoding for semi-stable extensions.

In fact, for an AF $F = (A, R)$ and $S \in cf(F)$ we need to check whether no $T \in cf(F)$ with $S_R^+ \subset T_R^+$ exists. Therefore we have to guess an arbitrary set T and saturate in case

(i) T is not conflict-free, and

(ii) $S_R^+ \not\subset T_R^+$.

The following module (together with π_{cf}) computes for a guessed subset $S \subseteq A$ the range S_R^+ of S in an AF $F = (A, R)$ (as introduced in Definition 5 on page 11).

$$\begin{aligned}\pi_{range} = \{\ &\text{in\_range}(X) \leftarrow \text{in}(X);\\ &\text{in\_range}(X) \leftarrow \text{in}(Y), \text{att}(Y, X);\\ &\text{not\_in\_range}(X) \leftarrow \text{arg}(X), \textit{not}\ \text{in\_range}(X)\ \}.\end{aligned}$$

In the next module we make a second guess for the set T. Then, $\text{in}/1$ holds the current guess for S and $\text{inN}/1$ holds the current guess for T.

$$\begin{aligned}\pi_{satstage} = \{\ &\text{inN}(X) \vee \text{outN}(X) \leftarrow \text{arg}(X); && (7.7)\\ &\text{fail} \leftarrow \text{inN}(X), \text{inN}(Y), \text{att}(X, Y); && (7.8)\\ &\text{fail} \leftarrow \text{eqplus}; && (7.9)\\ &\text{fail} \leftarrow \text{in\_range}(X), \text{not\_in\_rangeN}(X); && (7.10)\\ &\text{inN}(X) \leftarrow \text{fail}, \text{arg}(X); && (7.11)\\ &\text{outN}(X) \leftarrow \text{fail}, \text{arg}(X); && (7.12)\\ &\leftarrow \textit{not}\ \text{fail}\ \}. && (7.13)\end{aligned}$$

More specifically:

- In rule (7.7) we use disjunction for the guess. This is essential for the saturation technique because it allows for an argument a to have both $\text{inN}(a)$ and $\text{outN}(a)$ in the same answer-set which is not possible for the predicates $\text{in}/1$ and $\text{out}/1$ from module π_{cf}.

- Rule (7.8) checks requirement (i), so if the set T is not conflict-free we derive fail.

- Rule (7.9) fires in case $S_R^+ = T_R^+$ (indicated by predicate $\text{eqplus}/0$ described below).

- Rule (7.10) fires if there exists an $a \in S_R^+$ such that $a \notin T_R^+$ (here we use predicate $\text{in\_range}/1$ from above and predicate $\text{not\_in\_rangeN}/1$ which we also present below). As is easily checked one of the last two conditions holds exactly if (ii) holds.

- Next, the rules (7.11) and (7.12) saturate if fail was derived. This means that we derive for each $a \in A$ both $\text{inN}(a)$ and $\text{outN}(a)$ and therefore blow up the answer-sets.

- Finally, the constraint (7.13) rules out all guesses which do not contain fail.

To sum up, exactly those sets S survive where there is no T which is both conflict-free and has a bigger range than S.

In the module π_{rangeN} we compute the predicate $\mathrm{not\_in\_rangeN}/1$ via $\mathrm{undef\_upto}/2$. We use here the predicates $\inf/1$, $\mathrm{succ}/2$ and $\sup/1$ to compute the predicate $\mathrm{undef\_upto}(i, a)$ which states that the argument a is undefeated up to the i-th argument in the order $<$. Then, if an argument a is undefeated up to the supremum, we derive $\mathrm{not\_in\_rangeN}(a)$. Furthermore we compute the predicate $\mathrm{in\_rangeN}/1$ which gives us the range T_R^+ for the arguments in the second guess.

$$\begin{aligned}
\pi_{rangeN} = \{\ & \mathrm{undef_upto}(N, X) \leftarrow \mathrm{inf}(N), \mathrm{outN}(X), \mathrm{outN}(N);\\
& \mathrm{undef_upto}(N, X) \leftarrow \mathrm{inf}(N), \mathrm{outN}(X), \mathit{not}\ \mathrm{att}(N, X);\\
& \mathrm{undef_upto}(N, X) \leftarrow \mathrm{succ}(Z, N), \mathrm{undef_upto}(Z, X), \mathrm{outN}(N);\\
& \mathrm{undef_upto}(N, X) \leftarrow \mathrm{succ}(Z, N), \mathrm{undef_upto}(Z, X), \mathit{not}\ \mathrm{att}(N, X);\\
& \mathrm{not_in_rangeN}(X) \leftarrow \mathrm{sup}(M), \mathrm{outN}(X), \mathrm{undef_upto}(M, X);\\
& \mathrm{in_rangeN}(X) \leftarrow \mathrm{inN}(X);\\
& \mathrm{in_rangeN}(X) \leftarrow \mathrm{outN}(X), \mathrm{inN}(Y), \mathrm{att}(Y, X)\ \}.
\end{aligned}$$

In the module π_{eq}^+ we obtain eqplus, if the range from the first guess S and the second guess T is equal, i.e. if $S_R^+ = T_R^+$. This is done via the predicate $\mathrm{eqplus\_upto}/1$.

$$\begin{aligned}
\pi_{eq}^+ = \{\ & \mathrm{eqplus_upto}(X) \leftarrow \mathrm{inf}(X), \mathrm{in_range}(X), \mathrm{in_rangeN}(X);\\
& \mathrm{eqplus_upto}(X) \leftarrow \mathrm{inf}(X), \mathrm{not_in_range}(X), \mathrm{not_in_rangeN}(X);\\
& \mathrm{eqplus_upto}(X) \leftarrow \mathrm{succ}(Z, X), \mathrm{in_range}(X), \mathrm{in_rangeN}(X), \mathrm{eqplus_upto}(Z);\\
& \mathrm{eqplus_upto}(X) \leftarrow \mathrm{succ}(Y, X), \mathrm{not_in_range}(X), \mathrm{not_in_rangeN}(X),\\
& \qquad\qquad \mathrm{eqplus_upto}(Y);\\
& \mathrm{eqplus} \leftarrow \mathrm{sup}(X), \mathrm{eqplus_upto}(X)\ \};
\end{aligned}$$

We note here that both modules π_{rangeN} and π_{eq}^+ are stratified. Finally, we put everything together and obtain the encodings for stage semantics:

$$\pi_{stage} = \pi_{cf} \cup \pi_{<} \cup \pi_{range} \cup \pi_{rangeN} \cup \pi_{eq}^+ \cup \pi_{satstage}.$$

The following result gives the link between the stage extensions of an AF F and the answer-sets of the program π_{stage} with the input $\hat{F}$.

Proposition 15. *For any AF F, $stage(F) \cong \mathcal{AS}(\pi_{stage}(\hat{F}))$.*

The saturation encodings are quite complicated and normally one needs an ASP expert to design them. As many interesting problems require some kind of meta-reasoning, Gebser and Schaub designed the `metasp` optimization front end for the ASP-system `gringo/claspD` [70], which also allows for ASP beginners to encode problems which are on the second level of the polynomial hierarchy. In the next subsection we will explain how one can simplify the encodings of stage semantics using `metasp`.

`metasp` Encodings for Stage Semantics

In [52] the `metasp` approach has been used to simplify the encodings for preferred, semi-stable, stage and resolution-based grounded semantics. Here we picture this novel method by means of stage semantics. In particular we present the simplified encodings for stage semantics with the aid of the $\#minimize$ statement which are then evaluated with the subset-minimization semantics provided by `metasp`. For our encodings we do not need prioritization and weights, therefore these are omitted (i.e. set to default) in the minimization statements. The minimization technique is realized through meta programming techniques, which themselves are answer-set programs. This works as follows.

- The ASP encodings to solve are given to the grounder `gringo` which reifies the programs, i.e. outputs ground programs consisting of facts, which represent the rules and facts of the original input encodings.

- The grounder is then again executed on this output together with the meta programs which encode the optimization.

- Finally, `claspD` computes the answer-sets.

Note that here we use the version of `clasp` which supports disjunctive rules. Therefore for a program π and an AF F we have the following execution.

```
gringo -reify π(F̂) | \
    gringo - {meta.lp,metaO.lp,metaD.lp} \
    <(echo "optimize(1,1,incl).")  | claspD 0
```

Here, `meta.lp, metaO.lp` and `metaD.lp` are the encodings for the minimization statement. The statement `optimize(incl,1,1)` indicates that we use subset inclusion for the optimization technique using priority and weight 1.

Now the module for stage semantics is easy to encode using the minimization statement of `metasp`. Remember, a set S is a stage extension of an AF if it is conflict-free and has maximal range, so we minimize the predicate $\text{not\_in\_range}/1$ from the module π_{range}, and the encodings reduce to:

$$\pi_{stage\_metasp} = \pi_{cf} \cup \pi_{range} \cup \{\#minimize[\text{not\_in\_range}]\}.$$

The following result follows now directly.

Proposition 16. *For any AF F, we have $stage(F) \cong \mathcal{AS}(\pi_{stage\_metasp}(\hat{F}))$.*

Performance tests comparing the saturation encodings with the `metasp` encodings on different random instances showed that the use of this optimization front-end not only makes the encodings simpler but also faster. Especially in the case of stage semantics the runtime differences are in evidence. A detailed representation of the experimental evaluation can be found in [52].

Saturation Encodings for $stage2$ Semantics

Thanks to the modularity of ASP we can now more or less put the parts from computing $\Delta_{F,S}$, $[[F - \Delta_{F,S}]]$ and the encodings for stage extensions together. We just need some small modifications which we explain in the following. The saturation encodings for computing the $stage2$ extensions is composed as follows:

$$\pi_{stage2} = \pi_{cf} \cup \pi_{<} \cup \pi_{reach} \cup \pi_{inst} \cup \pi_{<'} \cup \pi_{range} \cup \pi^{+}_{eq'} \cup \pi_{rangeN'} \cup \pi_{satstage'}.$$

The four modules $\pi_{<'}$, $\pi^{+}_{eq'}$, $\pi_{rangeN'}$ and $\pi_{satstage'}$ require a slight modification from the original ones, namely $\pi_{<'}$ is defined as the order over the arguments contained in the instance $[[F - \Delta_{F,S}]]$.

$$\begin{aligned}
\pi_{<'} = \{\ & \mathrm{ltN}(X,Y) \leftarrow \mathrm{arg_new}(X), \mathrm{arg_new}(Y), X < Y;\\
& \mathrm{nsuccN}(X,Z) \leftarrow \mathrm{ltN}(X,Y), \mathrm{ltN}(Y,Z);\\
& \mathrm{succN}(X,Y) \leftarrow \mathrm{ltN}(X,Y), \mathit{not}\ \mathrm{nsuccN}(X,Y);\\
& \mathrm{ninfN}(Y) \leftarrow \mathrm{ltN}(X,Y);\\
& \mathrm{infN}(X) \leftarrow \mathrm{arg}(X), \mathit{not}\ \mathrm{ninfN}(X);\\
& \mathrm{nsupN}(X) \leftarrow \mathrm{ltN}(X,Y);\\
& \mathrm{supN}(X) \leftarrow \mathrm{arg_new}(X), \mathit{not}\ \mathrm{nsupN}(X)\ \}.
\end{aligned}$$

Then the modules $\pi^{+}_{eq'}$, $\pi_{rangeN'}$ and $\pi_{satstage'}$ use the predicates defined in $\pi_{<'}$, because the check if the guess is a stage extension is performed in the instance $[[F - \Delta_{F,S}]]$.

$$\begin{aligned}
\pi_{satstage'} = \{\ & \mathrm{inN}(X) \vee \mathrm{outN}(X) \leftarrow \mathrm{arg_new}(X);\\
& \mathrm{fail} \leftarrow \mathrm{inN}(X), \mathrm{inN}(Y), \mathrm{att_new}(X,Y);\\
& \mathrm{fail} \leftarrow \mathrm{eqplus};\\
& \mathrm{fail} \leftarrow \mathrm{in_range}(X), \mathrm{not_in_rangeN}(X);\\
& \mathrm{inN}(X) \leftarrow \mathrm{fail}, \mathrm{arg_new}(X);\\
& \mathrm{outN}(X) \leftarrow \mathrm{fail}, \mathrm{arg_new}(X);\\
& \leftarrow \mathit{not}\ \mathrm{fail}\ \}.
\end{aligned}$$

$$\begin{aligned}
\pi_{rangeN'} = \{\ & \mathrm{undef_upto}(N,X) \leftarrow \mathrm{infN}(N), \mathrm{outN}(X), \mathrm{outN}(N);\\
& \mathrm{undef_upto}(N,X) \leftarrow \mathrm{infN}(N), \mathrm{outN}(X), \mathit{not}\ \mathrm{att_new}(N,X);\\
& \mathrm{undef_upto}(N,X) \leftarrow \mathrm{succN}(Z,N), \mathrm{undef_upto}(Z,X), \mathrm{outN}(N);\\
& \mathrm{undef_upto}(N,X) \leftarrow \mathrm{succN}(Z,N), \mathrm{undef_upto}(Z,X), \mathit{not}\ \mathrm{att_new}(N,X);\\
& \mathrm{not_in_rangeN}(X) \leftarrow \mathrm{supN}(M), \mathrm{outN}(X), \mathrm{undef_upto}(M,X);\\
& \mathrm{in_rangeN}(X) \leftarrow \mathrm{inN}(X);\\
& \mathrm{in_rangeN}(X) \leftarrow \mathrm{outN}(X), \mathrm{inN}(Y), \mathrm{att_new}(Y,X)\ \}.
\end{aligned}$$

$$
\begin{aligned}
\pi^{+}_{eq'} = \{ \; & \text{eqplus\_upto}(X) \leftarrow \text{infN}(X), \text{in\_range}(X), \text{in\_rangeN}(X); \\
& \text{eqplus\_upto}(X) \leftarrow \text{infN}(X), \text{not\_in\_range}(X), \text{not\_in\_rangeN}(X); \\
& \text{eqplus\_upto}(X) \leftarrow \text{succN}(Z, X), \text{in\_range}(X), \text{in\_rangeN}(X), \text{eqplus\_upto}(Z); \\
& \text{eqplus\_upto}(X) \leftarrow \text{succN}(Y, X), \text{not\_in\_range}(X), \text{not\_in\_rangeN}(X), \\
& \qquad\qquad\qquad\quad \text{eqplus\_upto}(Y); \\
& \text{eqplus} \leftarrow \text{supN}(X), \text{eqplus\_upto}(X) \; \}.
\end{aligned}
$$

Finally we obtain the following result.

Proposition 17. *For any AF F, $stage2(F) \cong \mathcal{AS}(\pi_{stage2}(\hat{F}))$.*

`metasp` Encodings for $stage2$ Semantics

For $stage2$ semantics we can also make use of the `metasp` front end described above. Therefore we use the modules π_{cf}, $\pi_{<}$, π_{reach} and π_{inst} to compute for each guess the instance $[[F - \Delta_{F,S}]]$. For the check if the guessed set S is a stage extension of the instance, we first compute the predicate $\text{not\_in\_rangeN}/1$ via $\text{undef\_upto}/2$ in the slightly modified module $\pi_{rangeN''}$.

$$
\begin{aligned}
\pi_{rangeN''} = \{ \; & \text{undef\_upto}(N, X) \leftarrow \text{inf}(N), \text{out}(X), \text{out}(N); \\
& \text{undef\_upto}(N, X) \leftarrow \text{inf}(N), \text{out}(X), \textit{not } \text{att\_new}(N, X); \\
& \text{undef\_upto}(N, X) \leftarrow \text{succ}(Z, N), \text{undef\_upto}(Z, X), \text{out}(N); \\
& \text{undef\_upto}(N, X) \leftarrow \text{succ}(Z, N), \text{undef\_upto}(Z, X), \textit{not } \text{att\_new}(N, X); \\
& \text{not\_in\_rangeN}(X) \leftarrow \text{sup}(M), \text{out}(X), \text{undef\_upto}(M, X) \; \}.
\end{aligned}
$$

Then, we check if S is a naive extension of the instance in the module $\pi_{check\_naive'}$.

$$
\begin{aligned}
\pi_{check\_naive'} = \{ \; & \text{conflicting}(X) \leftarrow \text{att\_new}(Y, X), \text{out}(X), \text{in}(Y); \\
& \text{conflicting}(X) \leftarrow \text{att\_new}(X, Y), \text{out}(X), \text{in}(Y); \\
& \text{conflicting}(X) \leftarrow \text{att\_new}(X, X); \\
& \leftarrow \textit{not } \text{conflicting}(X), \text{not\_in\_rangeN}(X), \text{arg\_new}(X) \; \}.
\end{aligned}
$$

We put everything together including the minimize statement for $\text{not\_in\_rangeN}/1$.

$$
\begin{aligned}
\pi_{stage2\_metasp} = \; & \pi_{cf} \cup \pi_{<} \cup \pi_{reach} \cup \pi_{inst} \cup \pi_{check\_naive'} \cup \pi_{rangeN''} \\
& \cup \{\#\textit{minimize}[\text{not\_in\_range}]\}.
\end{aligned}
$$

Finally we obtain the following result.

Proposition 18. *For any AF F, $stage2(F) \cong \mathcal{AS}(\pi_{stage2\_metasp}(\hat{F}))$.*

7.2 Labelings

In the previous section we already performed a labeling when we computed the extensions of an AF. In the ASP encodings we "labeled" the arguments with "in" and "out". In this section we consider labelings as a direct approach to implement AFs as in [30, 97]. In the labeling-based approach one assigns each argument a label. Most commonly the arguments are labeled with *in*, *out* and *undec*, with the meaning that they are either accepted, rejected or one can not decide whether to accept or to reject the arguments. With this three-valued labels one obtains a more fine grained classification of the justification status of an argument.

There are definitions in terms of labelings for nearly all prominent semantics, for an overview we refer to [14], where also a labeling for $cf2$ semantics is included. However, we present here a slightly different definition of $cf2$ labelings because we belief that it reflects more the intuition of this semantics. Furthermore we also define $stage2$ labelings and we provide labeling based algorithms for $cf2$ and $stage2$ semantics, which are complexity sensitive in the sense that they reflect some results from Chapter 5.

In the following we introduce the necessary concepts for labelings and in particular for $cf2$ and $stage2$ labelings.

Definition 36. *Let $F = (A, R)$ be an AF. A* labeling *is a total function $\mathcal{L} : A \to \{in,\ out,\ undec\}$.*

Then, a labeling can be denoted as a triple $\mathcal{L} = (\mathcal{L}_{in}, \mathcal{L}_{out}, \mathcal{L}_{undec})$, where $\mathcal{L}_l = \{a \in A \mid \mathcal{L}(a) = l\}$. According to [14] conflict-free and stage labelings are given as follows.

Definition 37. *Let $F = (A, R)$ be an AF. Then, $\mathcal{L}$ is a* conflict-free labeling *of F, i. e. $\mathcal{L} \in cf_{\mathcal{L}}(F)$, iff*

- *for all $a \in \mathcal{L}_{in}$ there is no $b \in \mathcal{L}_{in}$ such that $(a, b) \in R$,*
- *for all $a \in \mathcal{L}_{out}$ there exists a $b \in \mathcal{L}_{in}$ such that $(b, a) \in R$*

Then, $\mathcal{L}$ is a stage labeling *of F, i. e. $\mathcal{L} \in stage_{\mathcal{L}}(F)$, iff $\mathcal{L} \in cf_{\mathcal{L}}(F)$ and there is no $\mathcal{L}' \in cf_{\mathcal{L}}(F)$ with $\mathcal{L}'_{undec} \subset \mathcal{L}_{undec}$.*

The following definition of a naive labeling slightly differs from the traditional definition, as there are no arguments labeled out. We need this special form of the naive labeling for the definition of the $cf2$ labeling.

Definition 38. *Let $F = (A, R)$ be an AF. Then, $\mathcal{L} \in naive_{\mathcal{L}}(F)$, iff*

- *for all $a \in \mathcal{L}_{in}$ there is no $b \in \mathcal{L}_{in}$ such that $(a, b) \in R$,*
- *$\mathcal{L}_{undec} = \{a \in A \setminus \mathcal{L}_{in}\}$ and $\mathcal{L}_{out} = \emptyset$,*
- *for all $a \in \mathcal{L}_{undec}$ there is an argument $b \in \mathcal{L}_{in}$, such that a is in conflict with b.*

Next, we define $cf2$ labelings, where an argument is labeled out iff it is attacked by an argument labeled in which does not belong to the same SCC.

Definition 39. *Let $F = (A, R)$ be an AF. Then, $\mathcal{L}$ is a $cf2$* labeling *of F, i.e. $\mathcal{L} \in cf2_{\mathcal{L}}(F)$, iff*

- *$\mathcal{L} \in naive_{\mathcal{L}}(F)$, in case $|SCCs(F) = 1|$.*

- *otherwise, $\forall C \in SCCs(F), \mathcal{L}|_{C \setminus D_F(\mathcal{L}_{in})} \in cf2_{\mathcal{L}}(F|_C - D_F(\mathcal{L}_{in}))$, and $\forall a \in A : a \in D_F(\mathcal{L}_{in}) \Leftrightarrow \mathcal{L}(a) = out$.*

It is easy to see that there is a one-to-one mapping between $cf2$ extensions and labelings, s.t. each extension S corresponds to a labeling $\mathcal{L}$ with $\mathcal{L}_{in} = S$ and $\mathcal{L}_{out} = \Delta_{F,S}$.

For $stage2$ labelings we use our alternative characterization.

Definition 40. *Let $F = (A, R)$ be an AF. Then, $\mathcal{L}$ is a $stage2$* labeling *of F, i. e. $\mathcal{L} \in stage2_{\mathcal{L}}(F)$, iff $\mathcal{L} \in cf_{\mathcal{L}}(F) \cap stage_{\mathcal{L}}([[F - \Delta_{F,\mathcal{L}_{in}}]])$, where $\Delta_{F,\mathcal{L}_{in}} \subseteq \mathcal{L}_{out}$.*

Again there is a one-to-one mapping between $stage2$ extensions and labelings, and each extension S corresponds to a labeling $\mathcal{L}$ with $\mathcal{L}_{in} = S$ and $\mathcal{L}_{out} = S^+ \setminus S$.

Labeling Algorithm for $cf2$

In the following we present a labeling-based algorithm which computes $cf2$-labelings/extensions. This algorithm is complexity-sensitive in the following sense. From Theorem 8 we know that on acyclic AFs, $cf2$ coincides with the grounded semantics and thus can be computed in polynomial time. To this end, the following algorithm is designed in the way that on acyclic AFs, there is no need for recursive calls. Notice that the other tractable fragments, i.e. symmetric and bipartite AFs, may propose an exponential number of extensions (the tractability for reasoning tasks was via some shortcut preventing us from computing all extensions) and thus not allow for an efficient computation of all extensions.

The following proposition identifies two rules to propagate already computed labels.

Proposition 19. *For AF $F = (A, R)$ and labeling $\mathcal{L} = (\mathcal{L}_{in}, \mathcal{L}_{out}, \mathcal{L}_{undec}) \in cf2_{\mathcal{L}}(F)$. Let $a \in A$, then $att(a) = \{b \in A \mid (b, a) \in R\}$ denotes all attackers of a.*

1. *For every $a \in A$: if $att(a) \subseteq \mathcal{L}_{out} \wedge (a, a) \notin R$ then $a \in \mathcal{L}_{in}$.*

2. *For every $a \in A$: if $\exists b \in \mathcal{L}_{in}, O \subseteq \mathcal{L}_{out} : (b, a) \in R \wedge a \not\Rightarrow_F^{A \setminus O} b$ then $a \in \mathcal{L}_{out}$.*

Proof. (1) As mentioned above $a \in \mathcal{L}_{out}$ iff $a \in \Delta_{F,\mathcal{L}_{in}}$. If all attackers of a are in $\Delta_{F,\mathcal{L}_{in}}$ we get that $\{a\}$ is an isolated argument in $[[F - \Delta_{F,S}]]$. Now, as $\mathcal{L} \in naive([[F - \Delta_{F,S}]])$ and $(a, a) \notin R$ we finally get $a \in \mathcal{L}_{in}$. (2) Using $\exists b \in \mathcal{L}_{in}, O \subseteq \mathcal{L}_{out} : (b, a) \in R \wedge a \not\Rightarrow_F^{A \setminus O} b$ and $O \subseteq \mathcal{L}_{out} = \Delta_{F,\mathcal{L}_{in}}$, we obtain that $\exists b \in \mathcal{L}_{in} : (b, a) \in R \wedge a \not\Rightarrow_F^{A \setminus \mathcal{L}_{out}} b$. As $\Delta_{F,\mathcal{L}_{in}}$ is a fixed-point we obtain that $a \in \Delta_{F,\mathcal{L}_{in}}$ and thus also $a \in \mathcal{L}_{out}$. $\square$

Algorithm 1 $\mathit{cf2}_{\mathcal{L}}(F, \mathcal{L})$

Require: AF $F = (A, R)$, labeling $\mathcal{L} = (\mathcal{L}_{in}, \mathcal{L}_{out}, \mathcal{L}_{undec})$;
Ensure: Return all *cf2* labelings of F.
1: $X = \{a \in \mathcal{L}_{undec} \mid att(a) \subseteq \mathcal{L}_{out}\}$;
2: $Y = \{a \in \mathcal{L}_{undec} \mid \exists b \in \mathcal{L}_{in}, (b, a) \in R, a \not\Rightarrow_F^{A \setminus \mathcal{L}_{out}} b\}$;
3: **while** $(X \cup Y) \neq \emptyset$ **do**
4: $\quad$ $\mathcal{L}_{in} = \mathcal{L}_{in} \cup X, \mathcal{L}_{out} = \mathcal{L}_{out} \cup Y, \mathcal{L}_{undec} = \mathcal{L}_{undec} \setminus (X \cup Y)$;
5: $\quad$ update X and Y;
6: **end while**
7: $B = \{a \in \mathcal{L}_{undec} \mid \mathcal{L}_{in} \cup \{a\} \in cf(F)\}$;
8: **if** $B \neq \emptyset$ **then**
9: $\quad$ $C = \{a \in B \mid \nexists b \in B : b \Rightarrow_F^{A \setminus \mathcal{L}_{out}} a, a \not\Rightarrow_F^{A \setminus \mathcal{L}_{out}} b\}$;
10: $\quad$ $\mathcal{E} = \emptyset$;
11: $\quad$ **for all** $\mathcal{L}' \in naive_{\mathcal{L}}(F|_C)$ **do**
12: $\quad\quad$ update $\mathcal{L}$ with $\mathcal{L}'$;
13: $\quad\quad$ $\mathcal{E} = \mathcal{E} \cup \mathit{cf2}_{\mathcal{L}}(F, \mathcal{L})$;
14: $\quad$ **end for**
15: $\quad$ **return** $\mathcal{E}$;
16: **else**
17: $\quad$ **return** $\{(\mathcal{L}_{in}, \mathcal{L}_{out}, \mathcal{L}_{undec})\}$;
18: **end if**

Description of Algorithm 1. The *cf2* labeling algorithm requires as input an AF $F = (A, R)$ and a labeling $\mathcal{L} = (\mathcal{L}_{in}, \mathcal{L}_{out}, \mathcal{L}_{undec})$. If $\mathit{cf2}_{\mathcal{L}}(F, \mathcal{L})$ is started with the initial labeling $\mathcal{L} = (\emptyset, \emptyset, A)$, it returns all *cf2* labelings of F.

- At the beginning, the two sets X and Y are computed. Where X identifies those arguments in $\mathcal{L}_{undec}$ which can directly be labeled with in, and Y identifies those arguments in $\mathcal{L}_{undec}$ which can directly be labeled with out according to Proposition 19. These new labeling modifications are performed in the "while-loop" till a fixed-point is reached.

- Next, the set B identifies all arguments which are labeled $undec$ and are not in conflict with the arguments in $\mathcal{L}_{in}$.

- Then, if $B \neq \emptyset$, the set C identifies the next SCCs to be labeled. Note here, C does not contain all arguments of an SCC, but all arguments which can be labeled in. To be more precise, self-attacking arguments are omitted in C.

- Next, in Line 11 a separated procedure identifies all naive labelings of the sub-framework $F|_C$. For each naive labeling $\mathcal{L}'$ we update the actual labeling $\mathcal{L}$ with $\mathcal{L}'$ and call $\mathit{cf2}_{\mathcal{L}}(F, \mathcal{L})$ recursively. Note, this step is a branch between different *cf2* extensions.

- Finally, the algorithm returns all *cf2* labelings of F.

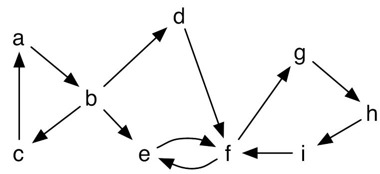

Figure 7.1: The argumentation framework F from Example 30.

Example 30. *Consider the AF from Example 5 as illustrated in Figure 7.1. We call* $cf2_{\mathcal{L}}(F, \mathcal{L})$ *with the initial labeling* $\mathcal{L} = (\emptyset, \emptyset, A)$.

At the beginning we have $X = \emptyset$, $Y = \emptyset$, $B = A$ *and* $C = \{a, b, c\}$. *We invoke the external procedure for computing the naive extensions of* $F|_C$ *which return three naive labelings* $\mathcal{L}_1 = (\{a\}, \emptyset, \{b, c\})$, $\mathcal{L}_2 = (\{b\}, \emptyset, \{a, c\})$ *and* $\mathcal{L}_3 = (\{c\}, \emptyset, \{a, b\})$. *For each of them the actual labeling is updated with* $\mathcal{L}' \in naive_{\mathcal{L}}(F|_C)$ *and* $cf2_{\mathcal{L}}(F, \mathcal{L})$ *is called.*

- *For* $\mathcal{L}_1$ *this looks as follows. We call* $cf2_{\mathcal{L}}(F, \mathcal{L})$ *with* $\mathcal{L} = (\{a\}, \emptyset, A \setminus \{a\})$. *Then,* $X = \emptyset$, $Y = \emptyset$, $B = \{d, e, f, g, h, i\}$ *and* $C = \{d\}$. *As* $F|_C$ *consists of the single argument d, we can update the actual labeling to* $(\{a, d\}, \emptyset, A \setminus \{a, d\})$ *and call* $cf2_{\mathcal{L}}$ *again.*
  - *Now,* $X = \emptyset$, $Y = \{f\}$ *and* $\mathcal{L}_{out} = \{f\}$. *Then,* $X = \{g\}$, $Y = \emptyset$ *and* $\mathcal{L}_{in} = \{a, d, g\}$. *Next,* $X = \emptyset$, $Y = \{h\}$ *and* $\mathcal{L}_{out} = \{f, h\}$. *Then,* $X = \{i\}$, $Y = \emptyset$ *and* $\mathcal{L}_{in} = \{a, d, g, i\}$. *Thus we obtain* $B = C = \{e\}$ *and we can update the labeling and return* $(\{a, d, e, g, i\}, \{f, h\}, \{b, c\})$.
- *For* $\mathcal{L}_2$ *we call* $cf2_{\mathcal{L}}(F, \mathcal{L})$ *with* $\mathcal{L} = (\{b\}, \emptyset, A \setminus \{b\})$. *Then,* $X = \emptyset$, $Y = \{d, e\}$ *and* $\mathcal{L}_{out} = \{d, e\}$. *Next,* $B = C = \{f, g, h, i\}$ *and as* $F|_C$ *has two naive extensions we can return the two cf2 labelings* $(\{b, f, h\}, \{d, e\}\{a, c, g, i\})$ *and* $(\{b, g, i\}, \{d, e\}, \{a, c, f, h\})$.
- *Finally for* $\mathcal{L}_3$ *we call* $cf2_{\mathcal{L}}(F, \mathcal{L})$ *with* $\mathcal{L} = (\{c\}, \emptyset, A \setminus \{c\})$. *Then,* $X = \emptyset$, $Y = \emptyset$, $B = \{d, e, f, g, h, i\}$ *and* $C = \{d\}$. *Here we have the same set B as in the step above for* $\mathcal{L}_1$, *which leads us to the cf2 labeling* $(\{c, d, e, g, i\}, \{f, h\}, \{a, b\})$.

$\diamond$

Labeling Algorithm for $stage2$

Now we give an algorithm for the computation of $stage2$ labelings. As the approach of $stage2$ is very close to the one of *cf2*, also the algorithm for $stage2$ labelings follows nearly the same procedure as Algorithm 1. In the following we first discuss the necessary modifications and then we present the algorithm, followed by an example where we explain step by step the procedure.

- As each $stage2$ extension is also a *cf2* extension we can apply Proposition 19 to $stage2$ as well, but we have to take into account the different definition of $\mathcal{L}_{out}$. To be more precise, for *cf2* labelings we have $\mathcal{L}_{out} = \Delta_{F,\mathcal{L}_{in}}$, whereas for $stage2$ labelings $\Delta_{F,\mathcal{L}_{in}} \subseteq \mathcal{L}_{out}$.

- Furthermore, we can not omit the self-loops in the restricted framework $F|_C$, as they are also necessary for the stage labelings. Thus we need to add them, which is done with the set D in Line 10 of Algorithm 2.
- Moreover, in Line 12 we have to replace $naive_{\mathcal{L}}(F|_C)$ by $stage_{\mathcal{L}}(F|_D)$.
- For the external procedure for stage labelings one can use the one presented in [28].

Algorithm 2 $stage2_{\mathcal{L}}(F, \mathcal{L})$

Require: AF $F = (A, R)$, labeling $\mathcal{L} = (\mathcal{L}_{in}, \mathcal{L}_{out}, \mathcal{L}_{undec})$;
Ensure: Return all $stage2$ labelings of F.
1: $X = \{a \in \mathcal{L}_{undec} \mid att(a) \subseteq \mathcal{L}_{out}\}$;
2: $Y = \{a \in \mathcal{L}_{undec} \mid \exists b \in \mathcal{L}_{in}, (b, a) \in R, a \not\Rightarrow_F^{A\setminus\mathcal{L}_{out}} b\}$;
3: **while** $(X \cup Y) \neq \emptyset$ **do**
4: $\quad$ $\mathcal{L}_{in} = \mathcal{L}_{in} \cup X, \mathcal{L}_{out} = \mathcal{L}_{out} \cup Y, \mathcal{L}_{undec} = \mathcal{L}_{undec} \setminus (X \cup Y)$;
5: $\quad$ update X and Y;
6: **end while**
7: $B = \{a \in \mathcal{L}_{undec} \mid \mathcal{L}_{in} \cup \{a\} \in cf(F)\}$;
8: **if** $B \neq \emptyset$ **then**
9: $\quad$ $C = \{a \in B \mid \nexists b \in B : b \Rightarrow_F^{A\setminus\mathcal{L}_{out}} a, a \not\Rightarrow_F^{A\setminus\mathcal{L}_{out}} b\}$;
10: $\quad$ $D = C \cup \{a \in \mathcal{L}_{undec} \mid \exists b \in C, a \Rightarrow_F^{A\setminus\mathcal{L}_{out}} b, b \Rightarrow_F^{A\setminus\mathcal{L}_{out}} a\}$
11: $\quad$ $\mathcal{E} = \emptyset$;
12: $\quad$ **for all** $\mathcal{L}' \in stage_{\mathcal{L}}(F|_D)$ **do**
13: $\quad\quad$ update $\mathcal{L}$ with $\mathcal{L}'$;
14: $\quad\quad$ $\mathcal{E} = \mathcal{E} \cup stage2_{\mathcal{L}}(F, \mathcal{L})$;
15: $\quad$ **end for**
16: $\quad$ **return** $\mathcal{E}$;
17: **else**
18: $\quad$ **return** $\{(\mathcal{L}_{in}, \mathcal{L}_{out}, \mathcal{L}_{undec})\}$;
19: **end if**

Example 31. *Consider the AF F pictured in Figure 7.2. We call $stage2_{\mathcal{L}}(F, \mathcal{L})$ with the initial labeling $\mathcal{L} = (\emptyset, \emptyset, A)$.*

We start with $X = \emptyset$, $Y = \emptyset$, $B = \{a, b, d, e, f, g, h, i\}$ and $C = \{a, b\}$. To complete the inner loop we compute $D = \{a, b, c\}$ which also takes the self-attacking argument c into account. Next we call the external procedure to obtain all stage labelings of the restricted AF $F|_D$ which gives us $\mathcal{L}_1 = (\{a\}, \{b\}, \{c\})$ and $\mathcal{L}_2 = (\{b\}, \{c\}, \{a\})$. Here we have the first branch where we update the actual labeling to the ones obtained from $stage_{\mathcal{L}}(F|_D)$.

- *For $\mathcal{L}_1$ we call $stage2_{\mathcal{L}}(F, \mathcal{L})$ with the updated labeling $\mathcal{L} = (\{a\}, \{b\}, A \setminus \{a, b\})$. This leads us to $X = \emptyset$, $Y = \emptyset$ and $B = C = D = \{d, e, f, g, h, i\}$. We call $stage_{\mathcal{L}}(F|_D)$ which returns $\mathcal{L}_{1,1} = (\{e, g, i\}, \{d, f, h\}, \emptyset)$ and $\mathcal{L}_{1,2} = (\{d, f, h\}, \{e, g, i\}, \emptyset)$ as the two stage labelings of $F|_D$. We update the actual labeling with them and branch another time.*

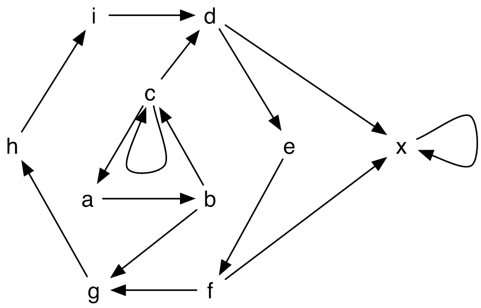

Figure 7.2: The argumentation framework F from Example 31.

  - *For* $\mathcal{L}_{1,1}$ *we call* $stage2_{\mathcal{L}}(F, \mathcal{L})$ *with* $\mathcal{L} = (\{a, e, g, i\}, \{b, d, f, h\}, \{c, x\})$*, where we have* $X = \emptyset$*,* $Y = \emptyset$ *and* $B = \emptyset$*. Thus, Algorithm 2 returns the* $stage2$ *labeling* $(\{a, e, g, i\}, \{b, d, f, h\}, \{c, x\})$.
  - *For* $\mathcal{L}_{1,2}$ *we call* $stage2_{\mathcal{L}}(F, \mathcal{L})$ *with* $\mathcal{L} = (\{a, d, f, h\}, \{b, e, g, i\}, \{c, x\})$*. Then,* $X = \emptyset$*,* $Y = \{x\}$ *and we obtain* $\mathcal{L}_{out} = \{b, e, g, i, x\}$*. As* $B = \emptyset$ *we return* $(\{a, d, f, h\}, \{b, e, g, i, x\}, \{c\})$.

- *For* $\mathcal{L}_2$ *we call* $stage2_{\mathcal{L}}(F, \mathcal{L})$ *with* $\mathcal{L} = (\{b\}, \{c\}, A \setminus \{b, c\})$*. Then* $X = \emptyset$*,* $Y = \{g\}$ *and* $\mathcal{L}_{out} = \{c, g\}$*. Next,* $X = \{h\}$*,* $Y = \emptyset$ *and* $\mathcal{L}_{in} = \{b, h\}$*. In the next iteration we have* $X = \emptyset$*,* $Y = \{i\}$ *and* $\mathcal{L}_{out} = \{c, g, i\}$ *and then* $X = \{d\}$*,* $Y = \emptyset$ *and* $\mathcal{L}_{in} = \{b, d, h\}$*. We continue with* $X = \emptyset$*,* $Y = \{e, x\}$ *and* $\mathcal{L}_{out} = \{c, e, g, i, x\}$ *and* $X = \{f\}$*,* $Y = \emptyset$ *and* $\mathcal{L}_{in} = \{b, d, f, h\}$*. Finally* $X = \emptyset$*,* $Y = \emptyset$ *and* $B = \emptyset$ *and the algorithm returns the last* $stage2$ *labeling of F, namely* $(\{b, d, f, h\}, \{c, e, g, i, x\}, \{a\})$.

$\diamond$

7.3 Web Application of ASPARTIX

As mentioned above, the ASP encodings described in Section 7.1 are fixed encodings, so one can use them together with the AF as input and the respective ASP-solver (`dlv`, `gringo/claspD`) without the need of any knowledge of ASP. However the user still needs to have an ASP-solver available and the encodings are only executable in the command line. To improve the usability we designed a web application of the system ASPARTIX which is freely accessible under `http://rull.dbai.tuwien.ac.at:8080/ASPARTIX/`. This tool uses the ASP encodings as described above and in [59] together with the ASP-solver `dlv`. However the actual usage is completely hidden from the user and thus makes the system easy to apply and understand. The advantages of this tool are the following.

- Many semantics are supported (admissible, stable, complete, grounded, preferred, semi-stable, ideal, *cf2*, stage, resolution-based grounded and $stage2$).

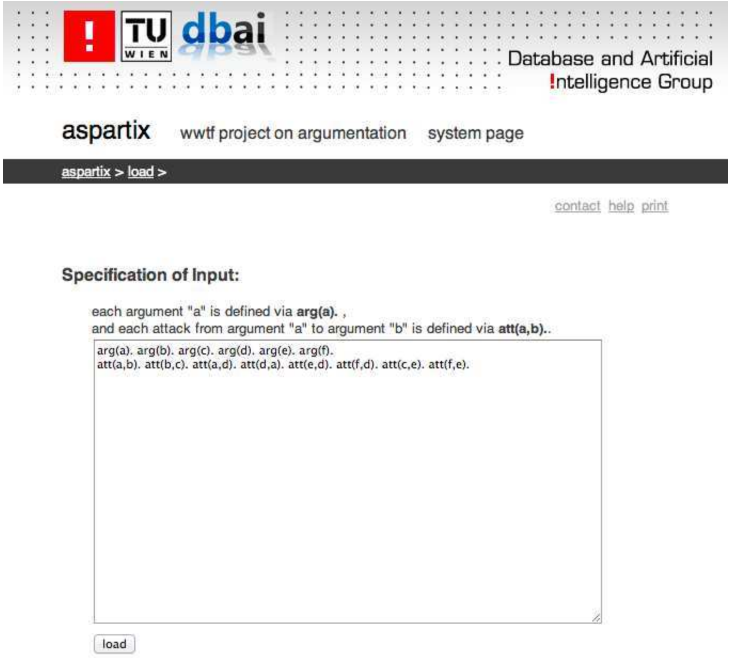

Figure 7.3: Input page of Web-ASPARTIX

- Compact syntax for input representation in terms of relational facts.
- Appealing graphical representation using the GraphMLViewer.
- Platform-independency and no installation is necessary.
- Runtimes scale up to frameworks with over 100 nodes.
- We can easily update the underlying engines (either ASPARTIX or DLV) to gain better performance or add to new semantics.

In Figures 7.3, 7.4 and 7.5 we pictured the input, the graph representation and the output of the web application of ASPARTIX where the input AF $F = (A, R)$ is defined with arguments $A = \{a, b, c, d, e, f\}$ and attacks $R = \{(a, b), (a, d), (b, c), (c, e), (d, a), (e, d), (f, d), (f, e)\}$. As the evaluation semantics we selected $cf2$.

7.4 Summary and Discussion

To sum up, we considered two distinct implementation methods. First the reduction-based approach with answer-set programming as the target formalism, second a direct approach where we used labeling-based algorithms to solve the respective reasoning tasks. In our case we mainly

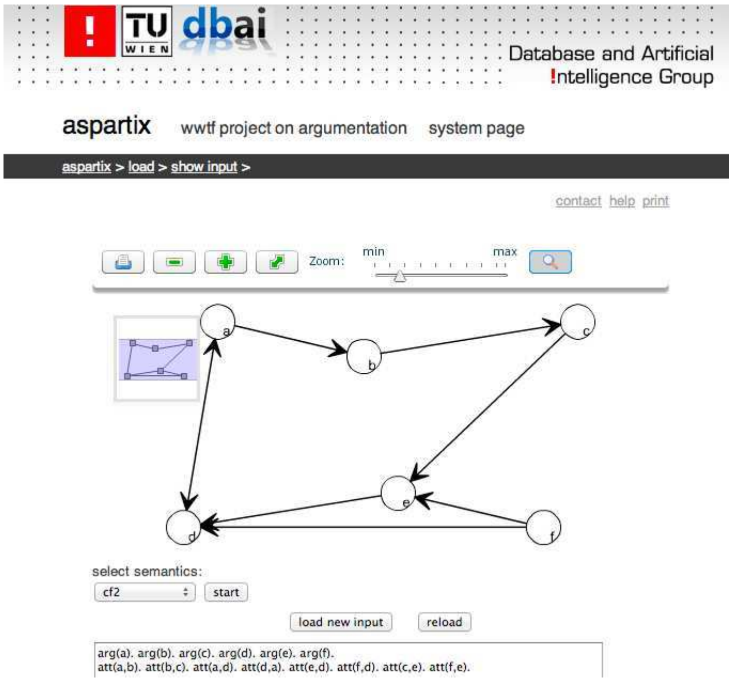

Figure 7.4: Graph representation and selection of semantics.

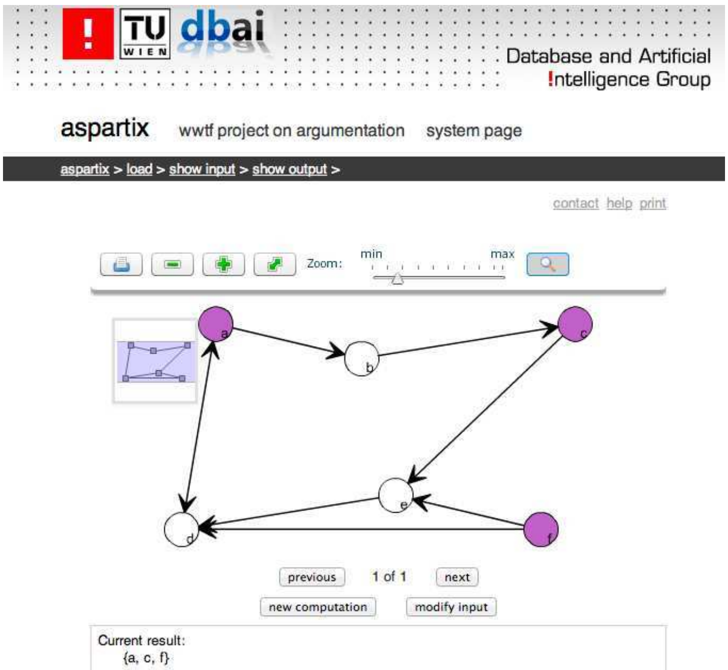

Figure 7.5: Output of extensions.

considered $cf2$ and $stage2$ semantics. On the ASP side we designed with the help of the alternative characterization of $cf2$ (resp. $stage2$) relatively compact encodings[11]. As the $stage2$ (resp. stage) semantics required the more involved concept of saturation encodings we made use of a novel optimization technique, the `metasp` encodings for the ASP-solver `gringo/claspD`. An experimental evaluation of the efficiency of the optimized encodings for preferred, semi-stable and stage semantics in [52] showed that the `metasp` frontend not only makes the encodings simpler but also has a significant improvement on the runtime.

The labeling-based algorithms for $cf2$ and $stage2$ are complexity sensitive in the sense that they do not perform recursive calls on acyclic AFs. Furthermore, we highlight that although the worst case runtime of the algorithms are exponential in the size of the AF, they are polynomial if one considers both the number of extensions and the size of the AF.

[11] All encodings are available online at `http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage/`.

CHAPTER 8

Conclusion

8.1 Summary

We shortly summarize the main results of this thesis. First, the alternative characterization of $cf2$ allowed to avoid the recursive computation of several sub-frameworks by shifting the recursion to the fixed-point operator $\Delta_{F,S}$. With this alternative characterization of $cf2$ several further investigation steps have been facilitated, like the complexity analysis, the investigation of equivalence and the implementation of this special semantics.

To overcome some of the shortcomings of $cf2$ we proposed to use the recursive schema of $cf2$ and instantiate the base case with stage semantics instead of only naive semantics. Thus, we obtained a new semantics which we called $stage2$. We showed that this novel semantics solves the problematic behavior of $cf2$ on longer cycles, in particular on cycles of length ≥ 6. Furthermore, $stage2$ satisfies directionality and weak reinstatement which was not the case for stage semantics. However, $stage2$ does not satisfy the weakest form of skepticism adequacy which is satisfied by $cf2$.

We provided the missing complexity results for $cf2$ and $stage2$ semantics. We summarize the obtained results for the standard reasoning problems for argumentation semantics and for the investigation of tractable fragments in Table 8.1. It turned out that both semantics are computationally hard and moreover, that $stage2$ is located on the second level of the polynomial hierarchy, thus it is among the hardest but also most expressiveness argumentation semantics. We were able to identify tractable fragments for both semantics, namely acyclic, bipartite and symmetric self-attack free frameworks.

The analysis of equivalence showed that for both, $cf2$ and $stage2$ semantics, strong equivalence coincides with syntactic equivalence. Hence, there are no redundant attacks at all. We made this behavior more explicit with the newly introduced succinctness property, which allows to relate the semantics according to how much meaning every attack has for the computation of the extensions. The succinctness property refers especially to semantics and not to specific frameworks. Thus, it can be seen as an additional possibility to compare argumentation seman-

	$cf2$	$stage2$
Ver_σ	in P	coNP-c
$Cred_\sigma$	NP-c	Σ_2^{P}-c
$Skept_\sigma$	coNP-c	Π_2^{P}-c
NE_σ	in P	in P
${Cred_\sigma}^{acycl}$	in P	in P
${Skept_\sigma}^{acycl}$	in P	in P
${Cred_\sigma}^{even-free}$	NP-c	coNP-h
${Skept_\sigma}^{even-free}$	coNP-c	coNP-h
${Cred_\sigma}^{bipart}$	in P	in P
${Skept_\sigma}^{bipart}$	in P	in P
${Cred_\sigma}^{sym}$	in P	in $\mathrm{P}/\Sigma_2^{\mathrm{P}}*$
${Skept_\sigma}^{sym}$	in P	in $\mathrm{P}/\Pi_2^{\mathrm{P}}*$

Table 8.1: Summary of complexity results.

tics. Furthermore we showed that stage semantics has the same kernel as stable semantics, thus strong equivalence for stable and stage semantics coincide.

In the implementation part we gave the ASP encodings of $cf2$ and $stage2$ semantics, where the alternative characterization facilitated this step. The encodings for $cf2$ only require the standard Guess&Check procedure for ASP programs. As $stage2$ is located at the second level of the polynomial hierarchy, we needed more involved programming techniques like the saturation encodings. To simplify those encodings we applied the novel `metasp` optimization front-end from the ASP system `gringo/claspD`. All these encodings are incorporated in the system ASPARTIX and available on the web. We also provided labeling based algorithms for $cf2$ and $stage2$ to directly compute the respective extensions. Finally, we illustrated the web application of ASPARTIX.

8.2 Critical Reflection

This thesis is dedicated to a comprehensive analysis of the $cf2$ semantics and one can conclude that this semantics is special in many different ways. Not only the special treatment of odd-length cycles but although the characterization requires more involved concepts, the computational complexity is not as hard as for preferred, stage or semi-stable semantics. Both, $cf2$ and $stage2$ satisfy the succinctness property, thus every attack and every argument has an influence in the computation of the extensions.

Amgoud and Vesic criticized in [3] that the notion of strong equivalence as introduced in

[84] is too strong and has no practical application at all. We do agree that for logic-based argumentation systems no self-attacking arguments do exist, but if one uses a different formalism for the instantiation process, like the ASPIC+ system [89] or ASP (as proposed by Dung in [37]), self-attacking arguments can occur. Therefore, knowing about redundant attacks for specific semantics, and the classification of them in terms of succinctness, is useful and can make the evaluation easier. As redundant attacks have no influence, they can be omitted already during the instantiation process which can be a useful simplification step.

In [14], there is a note that naive-based semantics may cause inconsistent conclusions when instantiated with the ASPIC+ schema as proposed in [29, 89]. This is also the case for $cf2$ semantics, where a counterexample against consistency has been discovered by Wolfgang Dvořák[12] and is explained in [14]. This does not mean that $cf2$ semantics yields inconsistent conclusions always, but that using the instantiation method from the ASPIC+ schema is not adequate for naive-based semantics.

In general the question which semantics is the best is hard to answer. The intention of this thesis was not to prove that $cf2$ or $stage2$ are better than other semantics, instead we wanted to provide an objective analysis of different computational and practical aspects. We belief that choosing the "right" semantics mainly depends on the particular application.

One important issue for the improvements of argumentation systems is the need for a benchmark library [53]. In [52] the runtime of some specific semantics has been evaluated on randomly generated instances. The $cf2$ semantics has not been included in this evaluation but we expect that the nature of this semantics may cause some performance loss as it is the case for the resolution-based grounded semantics. One crucial point in this context is that till now most of the systems have only been tested on randomly generated AFs because no real world examples are available. However, real problem instances may bear a specific structure which can have a significant effect on the runtime. Therefore, identifying those structures and taking them into account in the development of algorithms and systems is a very important topic which can cause significant improvements on the runtime behavior.

On the other hand, argumentation does serve as an application for other fields such as ASP or SAT solving. This is due to the fact that the representation of AFs as directed graphs is very simple but the complexity of the reasoning problems can be very hard. Thus, to use argumentation as benchmarks for these approaches leads to improvements of ASP solvers and subsequently to a better scalability of argumentation systems based on ASP, like ASPARTIX.

8.3 Related Work

In this section we discuss work which is related to the content of this thesis. As our work is dedicated to an analysis of the $cf2$ argumentation semantics, we start with related work on the *systematic evaluation of semantics*. Here one can mainly mention the work done by Baroni and Giacomin [7, 8, 10, 11]. They introduced several general evaluation criteria a semantics should fulfill. As none of the previously existing semantics satisfied all those criteria, they defined the resolution-based grounded semantics [9] which closed this gap. Also the $cf2$ semantics has been defined by them, to overcome the problems which arise on AFs with odd-length cycles.

[12] see `http://homepage.univie.ac.at/wolfgang.dvorak/files/cf2-challenge.pdf`

Caminada and Amgoud defined *rationality postulates* for argumentation systems [29]. In particular these postulates are defined for the ASPIC system [89] which is based on strict and defeasible rules. In contrast to the evaluation criteria proposed by Baroni and Giacomin, these postulates refer to the whole argumentation system and not to the individual semantics. We did not consider these postulates in more detail in this work because we concentrated on abstract argumentation only. Furthermore, the instantiation process in the ASPIC system is not adequate for naive-based semantics like $cf2$, because if one instantiates within this framework and then applies a semantics which is not admissible-based, the outcome turns out to be inconsistent.

Next, we consider different approaches which deal with the problematic of *cycles* in AFs, and in particular with odd-length cycles. Of course the $cf2$ semantics is not the only attempt to solve this problem. Bodanza and Tohmé introduced the *tolerant* semantics [23], which has the intuition that the defense of a set of arguments should not be defined in absolute terms but relative to other possible challenging sets of arguments. They propose an application of this semantics in the field of strategic argumentation games, where each player has to choose a set of arguments to confront with and defend against the possible choices of the other agent.

Gabbay introduced several *loop-busting* semantics in [64]. One of them, the LB2 semantics has shown to be equivalent to $cf2$. All these semantics are involved in the *equational approach* to argumentation networks [65]. The author also defines an equational approach to $stage2$ semantics in [64], namely LB$2 - stage$. Furthermore in [1] the authors propose the *Shkop* semantics which has been shown to be equivalent to LB4 from the loop-busting semantics in [64].

Roos proposed in [91] the *preferential model semantics* which also handles odd loops in a special way. The motivation for this semantics comes from the preferential model semantics for non-monotonic reasoning systems [73]. There, the attack relation is used to define preferences over states. So, not one argument is preferred over another one, but one prefers a state where the attacking argument is valid, over a state where the attacked argument is valid. This semantics results in different extensions than $cf2$, for example consider the AF $F = (A, R)$ with $A = \{a, b, c\}$ and $R = \{(a, b), (b, c), (c, a), (c, c)\}$. Then, $\{a\}$ and $\{b\}$ are $cf2$ extensions, but only $\{a\}$ is a *pm* extension, because the state a is preferred over the state b, as the only attacker of a is the argument c which is self-attacking.

Next, we consider work related to the investigation of equivalence as we did in Chapter 6. We start with Amgoud and Vesic who studied equivalence of *logic-based argumentation* [3] with respect to stable semantics. In particular the authors refined and extended the criteria from Oikarinen and Woltran for logic-based argumentation systems by taking the internal structure of the arguments into account.

Baumann characterized two new notions of equivalence, namely normal and strong expansion equivalence which lie in-between standard and strong equivalence [18]. There new arguments and attacks can be added with the condition that the attacks between the original arguments remain unchanged.

Cayrol et al. studied the *revision* of an argument system in [33] oriented on the field of belief revision [2]. There, the authors study the impact of adding a single new argument to an AF.

Finally we want to mention that the idea of considering strong equivalence of argumentation framework arose from the work on strong equivalence of logic programs [75, 100]. Therefore also the notion of strong equivalence for AFs is very similar to the one of logic programs.

Last, we say some words about related work on implementations. The only mentionable reference about a system supporting the *cf2* semantics is the work done by Osorio et al. [86]. They presented ASP encodings for the *cf2* semantics at COMMA 2010. Thus, at the same time as we presented the alternative characterization of *cf2* and the respective ASP encodings [67]. However, these encodings are not implemented in any system and as they are based on the original definition of *cf2* they are very hard to follow. Moreover, one can observe that disjunction has been used in some rule heads, thus they are not even adequate from a complexity point of view. As disjunctive logic programs have a data complexity of Σ_2^{P} (resp. Π_2^{P}), but the complexity of *cf2* is NP-complete (resp. coNP-complete).

Regarding other reductions from argumentation to logic programing one can mention the work of Nieves et al. [81, 82]. One aspect in their work is to use a fixed encoding *schema* to represent AFs as logic programs, and then show how different semantics for logic programs can be used to compute different forms of extensions using this particular schema. Most notably, they showed that in their setting the stable semantics (for logic programs) captures stable extensions of AFs, the well-founded semantics captures the grounded extension of AFs, and a novel stratification semantics [82] captures the *cf2* semantics. Osorio et al. [85] present an algorithm for computing preferred extensions (based on abductive logic programming) using a fixed logic program to characterize the admissible sets in the same manner as it is done in the ASPARTIX approach. In [81], a different approach to compute preferred extensions by means of logic programs has been proposed. However, this work requires a recompilation of the encoding for each particular AF. Similarly, Wakaki and Nitta [99] also provide ASP encodings for different semantics. In contrast to the ASPARTIX approach, their encodings for complete and stable semantics are based on labelings, whereas for grounded, preferred and semi-stable semantics they use a meta-programming technique applying additional translations for each AF into normal logic programs.

8.4 Future Work

In the following we will list some possible future directions. Regarding the alternative characterization, we note that it can also be seen as a general schema, where one can exchange the parts. For example, $sem(F) = \{S \mid \sigma(F) \cap \tau([[F - \Delta_{F,S}]])\}$, where for naive-based semantics $\sigma = naive$ and for admissible-based semantics $\sigma = adm$. One special case of this instantiation is $stable2(F) = \{S \mid S \in naive(F) \cap stable([[F - \Delta_{F,S}]])\}$ and it clearly holds that $stable2(F) = stable(F)$. The investigation of other such combinations might reveal new options.

As it turned out that strong equivalence is indeed very strong for many semantics, it can be beneficial to relax the notion of equivalence and for example consider a relativized notion, where source and target of attacks are restricted. This can be interesting in the course of two agents, where one can only point attacks from and to a specific set of arguments. Also a more fine grained classification of the semantics with respect to different notions of succinctness can be identified as a future direction. The information obtained there can help to improve instantiation methods.

Regarding implementations we would like to investigate if an optimization of the ASP encodings by using for example aggregates or symmetry breaking can improve the performance.

As far es we know there does not exist yet an appropriate instantiation method for naive-based semantics. It has been shown that both stage and $cf2$ semantics produce inconsistent solutions when instantiated within the ASPIC+ system. Thus, the identification of possible application scenarios for $cf2$ and $stage2$ semantics and the respective instantiation methods is still open.

With the use of the concept of Modular Logic Programming (MLP) [62] one can implement the whole argumentation process, from the instantiation of the arguments and attacks to the computation of the semantics. Due to the modularity of this approach, we plan to instantiate the frameworks from an input database and embed the existing ASP encodings in one program with several modules.

Bibliography

[1] Michael Abraham, Dov M. Gabbay, and Uri J. Schild. The handling of loops in talmudic logic, with application to odd and even loops in argumentation. In David Rydeheard, Andrei Voronkov, and Margarita Korovina, editors, *Proceedings of Higher-Order Workshop on Automated Runntime Verification and Debugging* (*HOWARD 60*), 2011.

[2] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *J. Symb. Log.*, 50(2):510–530, 1985.

[3] Leila Amgoud and Srdjan Vesic. On the equivalence of logic-based argumentation systems. In Salem Benferhat and John Grant, editors, *Proceedings of the 5th International Conference on Scalable Uncertainty Management (SUM 2011)*, volume 6929 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2011.

[4] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann Publishers, Inc., 1988.

[5] Katie Atkinson, Trevor J. M. Bench-Capon, and Sanjay Modgil. Argumentation for decision support. In Stéphane Bressan, Josef Küng, and Roland Wagner, editors, *Proceedings of the 17th International Conference on Database and Expert Systems Applications (DEXA 2006)*, volume 4080 of *Lecture Notes in Computer Science*, pages 822–831. Springer, 2006.

[6] Pietro Baroni and Massimiliano Giacomin. Solving semantic problems with odd-length cycles in argumentation. In Thomas D. Nielsen and Nevin L. Zhang, editors, *Proceedings of the 7th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2003)*, volume 2711 of *Lecture Notes in Computer Science*, pages 440–451. Springer, 2003.

[7] Pietro Baroni and Massimiliano Giacomin. Evaluating argumentation semantics with respect to skepticism adequacy. In Lluis Godo, editor, *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, (ECSQARU 2005)*, volume 3571 of *Lecture Notes in Computer Science*, pages 329–340. Springer, 2005.

[8] Pietro Baroni and Massimiliano Giacomin. On principle-based evaluation of extension-based argumentation semantics. *Artif. Intell.*, 171(10-15):675–700, 2007.

[9] Pietro Baroni and Massimiliano Giacomin. Resolution-based argumentation semantics. In Philippe Besnard, Sylvie Doutre, and Anthony Hunter, editors, *Proceedings of the 2nd Conference on Computational Models of Argument (COMMA 2008)*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, pages 25–36. IOS Press, 2008.

[10] Pietro Baroni and Massimiliano Giacomin. A systematic classification of argumentation frameworks where semantics agree. In Philippe Besnard, Sylvie Doutre, and Anthony Hunter, editors, *Proceedings of the 2nd International Conference on Computational Models of Argument, (COMMA 2008)*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, pages 37–48. IOS Press, 2008.

[11] Pietro Baroni and Massimiliano Giacomin. Semantics in abstract argumentation systems. In Iyad Rahwan and Guillermo R. Simari, editors, *Argumentation in Artificial Intelligence*, pages 25–44. Springer, 2009.

[12] Pietro Baroni, Massimiliano Giacomin, and Giovanni Guida. SCC-recursiveness: A general schema for argumentation Semantics. *Artif. Intell.*, 168(1–2):162–210, 2005.

[13] Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Giovanni Guida. Encompassing attacks to attacks in abstract argumentation frameworks. In Claudio Sossai and Gaetano Chemello, editors, *Proceedings of the 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2009)*, volume 5590 of *Lecture Notes in Computer Science*, pages 83–94, 2009.

[14] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *Knowledge Eng. Review*, 26(4):365–410, 2011.

[15] Pietro Baroni, Federico Cerutti, Paul E. Dunne, and Massimiliano Giacomin. Computing with infinite argumentation frameworks: The case of AFRAs. In Sanjay Modgil, Nir Oren, and Francesca Toni, editors, *Revised Selected Papers of the First International Workshop on Theorie and Applications of Formal Argumentation (TAFA 2011)*, volume 7132 of *Lecture Notes in Computer Science*, pages 197–214. Springer, 2011.

[16] Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Giovanni Guida. AFRA: Argumentation framework with recursive attacks. *Int. J. Approx. Reasoning*, 52(1):19–37, 2011.

[17] Pietro Baroni, Paul E. Dunne, and Massimiliano Giacomin. On the resolution-based family of abstract argumentation semantics and its grounded instance. *Artif. Intell.*, 175 (3–4):791–813, 2011.

[18] Ringo Baumann. Normal and strong expansion equivalence for argumentation frameworks. *Artif. Intell.*, 193:18–44, 2012.

[19] Trevor J. M. Bench-Capon. Value-based argumentation frameworks. In Salem Benferhat and Enrico Giunchiglia, editors, *Proceedings of the 9th International Workshop on Non-Monotonic Reasoning (NMR 2002)*, pages 443–454, 2002.

[20] Trevor J. M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *Journal of Logic and Computation*, 13(3):429–448, 2003.

[21] Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in artificial intelligence. *Artif. Intell.*, 171(10–15):619–641, 2007.

[22] Philippe Besnard and Anthony Hunter. *Elements of argumentation*. MIT Press, 2008.

[23] Gustavo A. Bodanza and Fernando A. Tohmé. Two approaches to the problems of self-attacking arguments and general odd-length cycles of attack. *Journal of Applied Logic*, 7 (4):403–420, 2009. Special Issue: Formal Models of Belief Change in Rational Agents.

[24] Francesco Calimeri, Giovambattista Ianni, Francesco Ricca, Mario Alviano, Annamaria Bria, Gelsomina Catalano, Susanna Cozza, Wolfgang Faber, Onofrio Febbraro, Nicola Leone, Marco Manna, Alessandra Martello, Claudio Panetta, Simona Perri, Kristian Reale, Maria C. Santoro, Marco Sirianni, Giorgio Terracina, and Pierfrancesco Veltri. The third answer set programming competition: Preliminary report of the system competition track. In James P. Delgrande and Wolfgang Faber, editors, *Logic Programming and Nonmonotonic Reasoning*, volume 6645 of *Lecture Notes in Computer Science*, pages 388–403. Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-20895-9_46.

[25] Martin Caminada. Semi-stable semantics. In Paul E. Dunne and Trevor J. M. Bench-Capon, editors, *Proceedings of the 1st Conference on Computational Models of Argument (COMMA 2006)*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 121–130. IOS Press, 2006.

[26] Martin Caminada. Comparing two unique extension semantics for formal argumentation: Ideal and eager. In *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2007)*, pages 81–87, 2007.

[27] Martin Caminada. An algorithm for computing semi-stable semantics. In Khaled Mellouli, editor, *Proceedings of the 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2007)*, volume 4724 of *Lecture Notes in Computer Science*, pages 222–234. Springer, 2007.

[28] Martin Caminada. An algorithm for stage semantics. In Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Guillermo R. Simari, editors, *Proceedings of the 3rd Conference on Computational Models of Argument (COMMA 2010)*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 147–158. IOS Press, 2010.

[29] Martin Caminada and Leila Amgoud. On the evaluation of argumentation formalisms. *Artif. Intell.*, 171(5–6):286–310, 2007.

[30] Martin Caminada and Dov M. Gabbay. A logical account of formal argumentation. *Studia Logica*, 93(2–3):109–145, 2009.

[31] Martin Caminada, Walter A. Carnielli, and Paul E. Dunne. Semi-stable semantics. *Journal of Logic and Computation*, 2011. doi: 10.1093/logcom/exr033.

[32] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. On the acceptability of arguments in bipolar argumentation frameworks. In Lluis Godo, editor, *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2005)*, volume 3571 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 2005.

[33] Claudette Cayrol, Florence Dupin de Saint Cyr-Bannay, and Marie-Christine Lagasquie-Schiex. Revision of an argumentation system. In Gerhard Brewka and Jérôme Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 124–134. AAAI Press, 2008.

[34] Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Symmetric argumentation frameworks. In Lluis Godo, editor, *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2005)*, volume 3571 of *Lecture Notes in Computer Science*, pages 317–328. Springer, 2005.

[35] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[36] Yannis Dimopoulos and Alberto Torres. Graph theoretical structures in logic programs and Default Theories. *Theor. Comput. Sci.*, 170(1–2):209–244, 1996.

[37] Phan M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.

[38] Phan M. Dung, Paolo Mancarella, and Francesca Toni. Computing ideal sceptical argumentation. *Artif. Intell.*, 171(10–15):642–674, 2007.

[39] Paul E. Dunne. Computational properties of argument systems satisfying graph-theoretic constraints. *Artif. Intell.*, 171(10–15):701–729, 2007.

[40] Paul E. Dunne and Trevor J. M. Bench-Capon. Complexity and combinatorial properties of argument systems. Technical report, Dept. of Computer Science, University of Liverpool, 2001.

[41] Paul E. Dunne and Trevor J. M. Bench-Capon. Coherence in finite argument systems. *Artif. Intell.*, 141(1/2):187–203, 2002.

[42] Paul E. Dunne and Martin Caminada. Computational complexity of semi-stable semantics in abstract argumentation frameworks. In Steffen Hölldobler, Carsten Lutz, and Heinrich

Wansing, editors, *Proceedings of the 11th European Conference on Logics in Artificial Intelligence (JELIA 2008)*, volume 5293 of *Lecture Notes in Computer Science*, pages 153–165. Springer, 2008.

[43] Paul E. Dunne and Michael Wooldridge. Complexity of abstract argumentation. In Iyad Rahwan and Guillermo R. Simari, editors, *Argumentation in Artificial Intelligence*, pages 85–104. Springer, 2009.

[44] Wolfgang Dvorák and Sarah A. Gaggl. Incorporating stage semantics in the scc-recursive schema for argumentation semantics. In *In Proceedings of the 14th International Workshop on Non-Monotonic Reasoning (NMR 2012)*, 2012.

[45] Wolfgang Dvorák and Sarah A. Gaggl. Computational aspects of cf2 and stage2 argumentation semantics. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the 4th International Conference on Computational Models of Argument (COMMA 2012)*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 273–284. IOS Press, 2012.

[46] Wolfgang Dvorák and Stefan Woltran. Complexity of semi-stable and stage semantics in argumentation frameworks. *Inf. Process. Lett.*, 110(11):425–430, 2010.

[47] Wolfgang Dvořák. *Computational Aspects of Abstract Argumentation*. PhD thesis, Vienna University of Technology, 2012.

[48] Wolfgang Dvořák and Stefan Woltran. Complexity of semi-stable and stage semantics in argumentation frameworks. *Inf. Process. Lett.*, 110(11):425–430, 2010. doi: 10.1016/j.ipl.2010.04.005.

[49] Wolfgang Dvořák and Stefan Woltran. On the intertranslatability of argumentation semantics. *J. Artif. Intell. Res. (JAIR)*, 41:445–475, 2011.

[50] Wolfgang Dvořák, Stefan Szeider, and Stefan Woltran. Reasoning in argumentation frameworks of bounded clique-width. In Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Guillermo R. Simari, editors, *Proceedings of the 3rd Conference on Computational Models of Argument (COMMA 2010)*, Frontiers in Artificial Intelligence and Applications, pages 219–230. IOS Press, 2010.

[51] Wolfgang Dvořák, Paul E. Dunne, and Stefan Woltran. Parametric properties of ideal semantics. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 851–856. AAAI Press, 2011.

[52] Wolfgang Dvořák, Sarah A. Gaggl, Johannes P. Wallner, and Stefan Woltran. Making use of advances in answer-set programming for abstract argumentation systems. *CoRR*, abs/1108.4942, 2011.

[53] Wolfgang Dvořák, Sarah A. Gaggl, Stefan Szeider, and Stefan Woltran. Benchmark libraries for argumentation. In Sascha Ossowski, editor, *Agreement Technologies*, volume 8 of *LGTS*, chapter The Added Value of Argumentation, pages 389–393. Springer, 2012.

[54] Wolfgang Dvořák, Sebastian Ordyniak, and Stefan Szeider. Augmenting tractable fragments of abstract argumentation. *Artificial Intelligence*, 186(0):157–173, 2012. ISSN 0004-3702. doi: 10.1016/j.artint.2012.03.002.

[55] Wolfgang Dvořák, Reinhard Pichler, and Stefan Woltran. Towards fixed-parameter tractable algorithms for abstract argumentation. *Artificial Intelligence*, 186(0):1 – 37, 2012. ISSN 0004-3702. doi: 10.1016/j.artint.2012.03.005.

[56] Wolfgang Dvořák, Stefan Szeider, and Stefan Woltran. Abstract argumentation via monadic second order logic. In Eyke Hüllermeier, Sebastian Link, Thomas Fober, and Bernhard Seeger, editors, *Scalable Uncertainty Management - 6th International Conference, SUM 2012, Marburg, Germany, September 17-19, 2012. Proceedings*, volume 7520 of *Lecture Notes in Computer Science*, pages 85–98. Springer, 2012.

[57] Uwe Egly, Sarah A. Gaggl, and Stefan Woltran. Aspartix: Implementing argumentation frameworks using answer-set programming. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Proceedings of the 24th International Conference on Logic Programming (ICLP 2008)*, volume 5366 of *Lecture Notes in Computer Science*, pages 734–738. Springer, 2008.

[58] Uwe Egly, Sarah A. Gaggl, and Stefan Woltran. Answer-set programming encodings for argumentation frameworks. In *1st Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2008)*, 2008.

[59] Uwe Egly, Sarah A. Gaggl, and Stefan Woltran. Answer-set programming encodings for argumentation frameworks. *Argument and Computation*, 1(2):144–177, 2010.

[60] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.*, 15(3–4):289–323, 1995.

[61] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/261124.261126.

[62] Thomas Eiter, Georg Gottlob, and Helmut Veith. Modular logic programming and generalized quantifiers. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 1997)*, volume 1265 of *Lecture Notes in Computer Science*, pages 290–309. Springer, 1997.

[63] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer, 2006.

[64] Dov M. Gabbay. The equational approach to cf2 semantics. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the 4th International Conference on Computational Models of Argument (COMMA 2012)*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 141–152. IOS Press, 2012.

[65] Dov M. Gabbay. Equational approach to argumentation networks. *Argument and Computation*, 3(2–3):87–142, 2012. doi: 10.1080/19462166.2012.704398.

[66] Sarah A. Gaggl. Towards a general argumentation system based on answer-set programming. In Manuel V. Hermenegildo and Torsten Schaub, editors, *Technical Communications of the 26th International Conference on Logic Programming (ICLP 2010)*, volume 7 of *Leibniz International Proceedings in Informatics*, pages 265–269. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[67] Sarah A. Gaggl and Stefan Woltran. cf2 semantics revisited. In Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Guillermo R. Simari, editors, *Proceedings of the 3rd Conference on Computational Models of Argument (COMMA 2010)*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 243–254. IOS Press, 2010.

[68] Sarah A. Gaggl and Stefan Woltran. Strong equivalence for argumentation semantics based on conflict-free sets. In Weiru Liu, editor, *Proceedings of the 11th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2011)*, volume 6717 of *Lecture Notes in Computer Science*, pages 38–49. Springer, 2011.

[69] Sarah A. Gaggl and Stefan Woltran. The cf2 argumentation semantics revisited. *Journal of Logic and Computation*, 2012. doi: 10.1093/logcom/exs011.

[70] Martin Gebser, Roland Kaminski, and Torsten Schaub. Complex optimization in answer set programming. *Theory and Practice of Logic Programming*, 11(4–5):821–839, 2011.

[71] Martin Gebser, Benjamin Kaufmann Roland Kaminski, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012. doi: 10.2200/S00457ED1V01Y201211AIM019.

[72] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3–4):365–386, 1991.

[73] Sarit Kraus, Daniel J. Lehmann, and Menachem Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artif. Intell.*, 44(1–2):167–207, 1990.

[74] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006. ISSN 1529-3785. doi: http://doi.acm.org/10.1145/1149114.1149117.

[75] Vladimir Lifschitz, David Pearce, and Agustín Valverde. Strongly equivalent logic programs. *ACM Trans. Comput. Log.*, 2(4):526–541, 2001.

[76] Nicolas Maudet, Simon Parsons, and Iyad Rahwan. Argumentation in multi-agent systems: Context and recent developments. In Nicolas Maudet, Simon Parsons, and Iyad Rahwan, editors, *Proceedings of the 3rd International Workshop on Argumentation in*

Multi-Agent Systems (ArgMAS 2006), volume 4766 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2006.

[77] Sanjay Modgil. Hierarchical argumentation. In Michael Fisher, Wiebe van der Hoek, Boris Konev, and Alexei Lisitsa, editors, *Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA 2006)*, pages 319–332. Springer, 2006.

[78] Sanjay Modgil. Reasoning about preferences in argumentation frameworks. *Artif. Intell.*, 173(9–10):901–934, 2009.

[79] Sanjay Modgil and Martin Caminada. Proof theories and algorithms for abstract argumentation frameworks. In Guillermo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 105–129. Springer, 2009. doi: 10.1007/978-0-387-98197-0_6.

[80] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics And Its Applications. OUP Oxford, 2006.

[81] Juan C. Nieves, Mauricio Osorio, and Ulises Cortés. Preferred extensions as stable models. *Theory and Practice of Logic Programming*, 8(4):527–543, 2008.

[82] Juan C. Nieves, Mauricio Osorio, and Claudia Zepeda. Expressing extension-based semantics based on stratified minimal models. In Hiroakira Ono, Makoto Kanazawa, and Ruy J. G. B. de Queiroz, editors, *Proceedings of the 16th International Workshop on Logic, Language, Information and Computation (WoLLIC 2009)*, volume 5514 of *Lecture Notes in Computer Science*, pages 305–319. Springer, 2009.

[83] Samer Nofal, Paul E. Dunne, and Katie Atkinson. On preferred extension enumeration in abstract argumentation. In Bart Verheij, Stefan Szeider, and Stefan Woltran, editors, *Proceedings of the 4th International Conference on Computational Models of Argument (COMMA 2012)*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 205–216. IOS Press, 2012.

[84] Emilia Oikarinen and Stefan Woltran. Characterizing strong equivalence for argumentation frameworks. *Artif. Intell.*, 175(14–15):1985–2009, 2011.

[85] Mauricio Osorio, Claudia Zepeda, Juan C. Nieves, and Ulises Cortés. Inferring acceptable arguments with answer set programming. In *Proceedings of the 6th Mexican International Conference on Computer Science (ENC 2005)*, pages 198–205. IEEE Computer Society, 2005.

[86] Mauricio Osorio, Juan C. Nieves, and Ignasi Gómez-Sebastià. CF2-extensions as answer-set models. In Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Guillermo R. Simari, editors, *Proceedings of the 3rd Conference on Computational Models of Argument (COMMA 2010)*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 391–402. IOS Press, 2010.

[87] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[88] John L. Pollock. Justification and defeat. *Artif. Intell.*, 67(2):377–407, 1994.

[89] Henry Prakken. An abstract framework for argumentation with structured arguments. *Argument and Computation*, 1(2):93–124, 2010.

[90] Iyad Rahwan and Guillermo R. Simari. *Argumentation in Artificial Intelligence*. Springer, 2009.

[91] Nico Roos. The relation between preferential model and argumentation semantics. In *In Proceedings of the 13th International Workshop on Non-Monotonic Reasoning (NMR 2010)*, 2010.

[92] Nicolás D. Rotstein, Martín O. Maguillansky, Alejandro J. García, and Guillermo R. Simari. An abstract argumentation framework for handling dynamics. In *Proceedings of the 12th International Workshop on Non-Monotonic Reasoning (NMR 2008)*, pages 131–139, September 2008.

[93] Nicolás D. Rotstein, Martín O. Moguillansky, Marcelo A. Falappa, Alejandro J. García, and Guillermo R. Simari. Argument theory change: Revision upon warrant. In Philippe Besnard, Sylvie Doutre, and Anthony Hunter, editors, *Proceedings of the 2nd Conference on Computational Models of Argument (COMMA 2008)*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, pages 336–347. IOS Press, 2008.

[94] Robert E. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1 (2):146–160, 1972.

[95] Francesca Toni and Marek Sergot. Argumentation and answer set programming. In Marcello Balduccini and Tran C. Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, volume 6565 of *Lecture Notes in Computer Science*, pages 164–180. Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-20832-4_11.

[96] Bart Verheij. Two approaches to dialectical argumentation: Admissible sets and argumentation stages. In John-Jules Ch. Meyer and Linda C. van der Gaag, editors, *Proceedings of the Eighth Dutch Conference on Artificial Intelligence (NAIC 1996)*, pages 357–368. University of Utrecht, 1996.

[97] Bart Verheij. A labeling approach to the computation of credulous acceptance in argumentation. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 623–628, 2007.

[98] Gerard Vreeswijk and Henry Prakken. Credulous and sceptical argument games for preferred semantics. In Manuel Ojeda-Aciego, Inman P. de Guzmán, Gerhard Brewka, and Luís M. Pereira, editors, *Proceedings of the European Workshop on Logics in Artificial Intelligence (JELIA 2000)*, volume 1919 of *Lecture Notes in Computer Science*, pages 239–253. Springer, 2000.

[99] Toshiko Wakaki and Katsumi Nitta. Computing argumentation semantics in answer set programming. In Hiromitsu Hattori, Takahiro Kawamura, Tsuyoshi Idé, Makoto Yokoo, and Yohei Murakami, editors, *New Frontiers in Artificial Intelligence (JSAI 2008), Conference and Workshops*, volume 5447 of *Lecture Notes in Computer Science*, pages 254–269. Springer, 2008.

[100] Stefan Woltran. A common view on strong, uniform, and other notions of equivalence in answer-set programming. *Theory and Practice of Logic Programming*, 8(2):217–234, 2008.

[101] Adam Z. Wyner, Trevor J. M. Bench-Capon, and Katie Atkinson. Towards formalising argumentation about legal cases. In Kevin D. Ashley and Tom M. van Engers, editors, *Proceedings of the 13th International Conference on Artificial Intelligence and Law (ICAIL)*, pages 1–10. ACM, 2011.

APPENDIX A

Curriculum Vitae

Curriculum Vitae

Sarah Alice Gaggl

Address

Institute of Information Systems
Database and Artificial Intelligence Group
Vienna University of Technology
Favoritenstraße 9
A-1040 Wien
Austria
Phone: +43-1-58801-18438
Fax: +43-1-58801-18492
Email: gaggl@dbai.tuwien.ac.at
Homepage: www.dbai.tuwien.ac.at/staff/gaggl/

Personal Details

Gender: Female
Date of birth: 22th of February, 1980
Place of birth: Villach, Austria
Present Citizenship: Austria

Education

08/2010	22nd European Summer School in Logic, Language and Information (ESSLLI 2010), Denmark, Copenhagen, August 9-20, 2010.
08/2009	Advanced Course in Artificial Intelligence (ACAI 2009), GB, Belfast, August 23-29, 2009.
07/2009	21st European Summer School in Logic, Language and Information (ESSLLI 2009), France, Bordeaux, July 20-31, 2009.
Since 04/2009	PhD student at the Vienna University of Technology.
2001–2009	Student of Computer Science at the Vienna University of Technology; graduation as a Bachelor of Science (BSc) in *Medicine and Computer Science*, and graduation as a Master of Science (MSc) in *Computational Intelligence* with distinction. Thesis: *Solving Argumentation Frameworks using Answer Set Programming*; Supervisor: Ao.Univ.Prof. Dr. Uwe Egly.

Working Experience and Reviewing

04/2009-09/2012 Research Assistant at the *Database and Artificial Intelligence Group* of the *Institute of Information Systems* at the Vienna University of Technology.

Project title: *New Methods for Analyzing, Comparing, and Solving Argumentation Problems*; Supervisor: Privatdoz. Dr. Stefan Woltran.
Supported by the WWTF under grant ICT 08-028.

- Reviewing for Journal of Logic and Computation, Special Issue on 20 years of Argument-based Inference (JLCabi 2011).
- Reviewing for international conferences and workshops including: 20th European Conference on Artificial Intelligence (ECAI 2012), 8th Reasoning Web Summer School 2012, 4th International Conference on Agents and Artificial Intelligence (ICAART 2012), 25th Conference on Artificial Intelligence (AAAI 2011), 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2011), 1st International Workshop on the Theory and Applications of Formal Argumentation (TAFA 2011), 19th European Conference on Artificial Intelligence (ECAI 2010), 20th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux 2011)

Awards

06/2012 *Best Student Paper Prize* at the 14th International Workshop on Non-Monotonic Reasoning (NMR 2012) for the article *Incorporating Stage Semantics in the SCC-recursive Schema for Argumentation Semantics*, joint work with Wolfgang Dvořák.

09/2010 *Best Student Paper Award* at the International Conference on Computational Models of Argument (COMMA 2010) for the paper *cf2 Semantics Revisited*.

Research Visits

03/2011 Prof. Ken Satoh, National Institute of Informatics (NII), Tokyo, Japan.

12/2009 Group of Prof. Gerd Brewka, Univ. Leipzig, Germany.

Grants and Internships

- COST Travel Grant for attending the Doctoral Consortium at KR 2012.
- COST Travel Award for attending the London Argumentation Forum (LAF) in April 2012.
- IJCAI Travel Grant for attending IJCAI 2011.
- International Internship at National Institute of Informatics (NII), Tokyo, Japan, 2011.
- ECCAI Travel Award for attending ACAI 2009.

Invited Talks and Presentations

- *Incorporating the Stage Semantics in the SCC-recursive Schema for Argumentation Semantics*. London Argumentation Forum (LAF). King's College, London, April 20, 2012.
- *Making Use of Advances in Answer-Set Programming for Abstract Argumentation Systems*. Computational Logic and Knowledge Representation Workshop. Invited Talk. UPS University, IRIT, Toulouse, France, October 21-22, 2011.
- *Strong Equivalence for Argumentation Semantics based on Conflict-free Sets*. Argumentation Christmas Meeting. Presentation. Vienna University of Technology, Austria, December 7-8, 2010.

Teaching Experience

At the Vienna University of Technology.

- Seminar "Logic Seminar", (3.0 ECTS), summer term (ST) 2012.
- Exercises for the course "Introduction to Knowledge-based Systems", (5.0 ECTS), ST 2012.
- Exercises for the course "Introduction to Artificial Intelligence", (3.0 ECTS), ST 2012.
- Course "Abstract Argumentation", (4.5 ECTS), winter term (WT) 2011/12.
- Teaching Assistant for Laboratory Exercise "Introduction to Knowledge-based Systems", (1.5 ECTS), WT 2008/09.
- Teaching Assistant for Laboratory Exercise "Logic-oriented Programming", (3.0 ECTS), ST 2008.
- Teaching Assistant for Lecture and Exercise "Fundamentals of Computer Science", (6.0 ECTS), ST 2002–2008, WT 2005/06–2008/09.

Publications

[16] W. Dvořák, S. A. Gaggl. Computational Aspects of cf2 and stage2 Argumentation Semantics. In *Proceedings of the 4th International Conference on Computational Models of Argument* (COMMA 2012), Vienna, Austria, September 10-12, 2012, volume 245 of Frontiers in Artificial Intelligence and Applications, pages 273-284. IOS Press, 2012.

[15] W. Dvořák, S. A. Gaggl. Incorporating Stage Semantics in the SCC-recursive Schema for Argumentation Semantics. In *Proceedings of the 14th International Workshop on Non-Monotonic Reasoning* (NMR 2012), Rome, Italy, June 8-10, 2012.

[14] S. A. Gaggl, S. Woltran. The cf2 Argumentation Semantics Revisited. In *Journal of Logic and Computation* 2012; doi: 10.1093/logcom/exs011.

[13] W. Dvořák, S. A. Gaggl. Incorporating Stage Semantics in the SCC-recursive Schema for Argumentation Semantics. Technical Report DBAI-TR-2012-78, Technische Universität Wien, 2012.

[12] S. A. Gaggl, S. Woltran. The cf2 Argumentation Semantics Revisited. Technical Report DBAI-TR-2012-77, Technische Universität Wien, 2012.

[11] W. Dvořák, S. A. Gaggl, J. P. Wallner, S. Woltran. Making Use of Advances in Answer-Set Programming for Abstract Argumentation Systems. In *Proceedings of 19th International Conference on Applications of Declarative Programming and Knowledge Management* (INAP 2011), Vienna, Austria, 2011.

[10] W. Dvořák, S. A. Gaggl, J. P. Wallner, S. Woltran. Making Use of Advances in Answer-Set Programming for Abstract Argumentation Systems. Technical Report DBAI-TR-2011-70, Technische Universität Wien, 2011.

[9] S. A. Gaggl, S. Woltran. Strong Equivalence for Argumentation Semantics based on Conflict-free Sets. In *Proceedings of 11th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty* (ECSQARU 2011), Belfast, Ireland, pages 38–49, 2011.

[8] S. A. Gaggl, S. Woltran. Strong Equivalence for Argumentation Semantics based on Conflict-free Sets. Sarah Gaggl, and Stefan Woltran. Technical Report DBAI-TR-2011-68, Technische Universität Wien, 2011.

[7] S. A. Gaggl, S. Woltran. cf2 Semantics Revisited. In *Proceedings of the Third International Conference on Computational Models of Argument (COMMA 2010), Desenzano del Garda, Italy*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 243–254. IOS Press, 2010.

[6] S. A. Gaggl. Towards a General Argumentation System based on Answer-Set Programming. In *ICLP (Technical Communications)* 2010: 265–269.

[5] U. Egly, S. A. Gaggl, S. Woltran. Answer-Set Programming Encodings for Argumentation Frameworks. In *Argument and Computation*, 1(2): 147–177 (2010).

[4] S. A. Gaggl. ASPARTIX: A System for Computing Different Argumentation Semantics in Answer-Set Programming; *Advanced Course in Artificial Intelligence* (ACAI 2009), Belfast, Ireland; Poster.

[3] U. Egly, S. A. Gaggl, S. Woltran. ASPARTIX: Implementing Argumentation Frameworks Using Answer-Set Programming. *Proceedings of the 24th International Conference on Logic Programming* (ICLP 2008), Udine, Italy pages 734-738. Springer LNCS 5366, 2008.

[2] U. Egly, S. A. Gaggl, S. Woltran. Answer-Set Programming Encodings for Argumentation Frameworks. *1st Workshop on Answer Set Programming and other Computing Paradigms* (ASPOCP 2008), Udine, Italy, 2008.

[1] U. Egly, S. A. Gaggl, S. Woltran. Answer-Set Programming Encodings for Argumentation Frameworks. Technical Report DBAI-TR-2008-62, Technische Universität Wien, 2008.

Language Knowledge

German	native
English	very good
Spanish	excellent

character:

Vienna, February 15, 2013